data mark

3 subject notebook

120 SHEETS
11 IN. X 8½ IN.

3 SECTIONS/**40 SHEETS EACH**

COLLEGE RULED

#88407

DATACOM, INC • WILLOW GROVE, PA 19090

TIMEX FISCAL WEEK CALENDAR

1983

JANUARY

S	M	T	W	T	F	S	FW
						1	
2	3	4	5	6	7	8	1
9	10	11	12	13	14	15	2
16	17	18	19	20	21	22	3
23	24	25	26	27	28	29	4
30	31						5

FEBRUARY

S	M	T	W	T	F	S	FW
		1	2	3	4	5	
6	7	8	9	10	11	12	6
13	14	15	16	17	18	19	7
20	21	22	23	24	25	26	8
27	28						9

MARCH

S	M	T	W	T	F	S	FW
		1	2	3	4	5	
6	7	8	9	10	11	12	10
13	14	15	16	17	18	19	11
20	21	22	23	24	25	26	12
27	28	29	30	31			13

APRIL

S	M	T	W	T	F	S	FW
					1	2	
3	4	5	6	7	8	9	14
10	11	12	13	14	15	16	15
17	18	19	20	21	22	23	16
24	25	26	27	28	29	30	17

MAY

S	M	T	W	T	F	S	FW
1	2	3	4	5	6	7	18
8	9	10	11	12	13	14	19
15	16	17	18	19	20	21	20
22	23	24	25	26	27	28	21
29	30	31					22

JUNE

S	M	T	W	T	F	S	FW
			1	2	3	4	
5	6	7	8	9	10	11	23
12	13	14	15	16	17	18	24
19	20	21	22	23	24	25	25
26	27	28	29	30			26

JULY

S	M	T	W	T	F	S	FW
					1	2	
3	4	5	6	7	8	9	27
10	11	12	13	14	15	16	28
17	18	19	20	21	22	23	29
24	25	26	27	28	29	30	30
31							

AUGUST

S	M	T	W	T	F	S	FW
	1	2	3	4	5	6	31
7	8	9	10	11	12	13	32
14	15	16	17	18	19	20	33
21	22	23	24	25	26	27	34
28	29	30	31				35

SEPTEMBER

S	M	T	W	T	F	S	FW
				1	2	3	
4	5	6	7	8	9	10	36
11	12	13	14	15	16	17	37
18	19	20	21	22	23	24	38
25	26	27	28	29	30		39

OCTOBER

S	M	T	W	T	F	S	FW
						1	
2	3	4	5	6	7	8	40
9	10	11	12	13	14	15	41
16	17	18	19	20	21	22	42
23	24	25	26	27	28	29	43
30	31						44

NOVEMBER

S	M	T	W	T	F	S	FW
		1	2	3	4	5	
6	7	8	9	10	11	12	45
13	14	15	16	17	18	19	46
20	21	22	23	24	25	26	47
27	28	29	30				48

DECEMBER

S	M	T	W	T	F	S	FW
				1	2	3	
4	5	6	7	8	9	10	49
11	12	13	14	15	16	17	50
18	19	20	21	22	23	24	51
25	26	27	28	29	30	31	52

XFER bytes from bank 0 to bank 255 or LDIR (would have to do BANK_ENABLE).

Initially the 5 overhead bytes at 8000h verify AROS and language is BASIC

Start address is beginning of program line and part of first block moved to E_LINE.

Why not take all new channel tales and put that info into SysCon? Just do one XFER of all overhead info into SYSCON. Have a search routine for that returns pointing to first byte of record. The routine that called it can then add an offset to get to the element it wants.

OK.

Check more thoroughly. May not work with stream data. Perhaps first byte should be 128 base.

To do select we get offset from stream data from CHINIT.

Maybe we can make STRMS to be 2 byte data but still use low byte for internal devices. Anything above 8000h indicates internal device and the 8000h will be subtracted to find offset from internal table.

For initialization of BEU devices

- Write something to 0000
- Read it back
- If FF then there is no device else if different from what was written then it is an intelligent device ROM else it is RAM

RAM should go into SYSCON.

BASIC progs in AROS must end with 80H.

Maybe presence flag should indicate:

0 = nothing

2 = RAM

1 = intelligent device

Let's put BANK_NO into SYSCON

PRINT #, INPUT # must work for BEU devices.

- F.S. - bug fixes
- $DD - Basic AROS handling
  bug fixes
  new commands

Do BASIC AROS handling for both architecture and design specs.

Changes for FS and architecture by 4/15.

Handle implementation ~~of RESET~~ and BASIC AROS (READ cmd)

- Initialization/channels by 4/15
- AROS by 4/22

Check INPUT # and PRINT # for intelligent devices

Need a CURCBN - current channel bank #.

In LSC2 (p. 9 EDIT) check CURCBN and if in ~~2-254~~ 1-253 do a CALL_BANK instead of CALL JP_HL. This covers both INCH and WRCH.

~~check out scroll~~

Initialization can't use BANK_ENABLE routine until after SYSCON is built.

To jump over to ext. bank write 80H to port FF and ~~call at~~ jump to directly.

~~Since we don't have Ext ROM yet, replace the GOTO_BANK with a JP E000~~

~~Ornstein wants 2 flags. Bits 5,6,7 appear to be open in FLAGS2.~~

~~We can also use bit 6 or 7 of TVFLAG for the VIDMODE flag.~~

If LROS found, don't check for AROS.

RESET * is system cold start of all devices.

If BASIC AROS is stopped, normal mode is entered but GOTOs etc will still work on AROS.

Ornstein wants to know how much ram the 2000 additions consume.

Check GOTO_USR routine from AROS. Where does it call - in AROS or HOME ROM.

Call this procedure RESALL. Close everything that's open. The CLOSE routine expects a stream # on calculator stack.

NO - this might attempt to call a close routine that no longer exists.
SO - just zero out the stream data.

BLDSCT should mark as not present any banks that aren't present. Put RSTSTR label at CLOSE+6 (LD BC,0)

Move SYSCON to 6000h - 171d (0ABh).

To test bank switching, design program to handle all configurations and do all tests. Will make determination based on SYSCON. For exp. banks, will look for RAM to XFER test code to exp. bank.

Init tests XFER from dock/exp to home.

For RESSCT

~~1. Read dev spec of each currently existing bank. If same as in SYSCON for that bank, go on to next bank.~~

Write reset cmd while read status returns banks responding do:

- ~~XFER dev spec~~
- ~~Search SYSCON for spec~~
- ~~If found, mark presence~~
- Test for RAM

Make a spare entry in SYSCON for a temp. Basically run thru BLDSCR but instead of read read to temp entry and don't do WRITE_BS_REG stuff.

Before writing init_done, do search existing entries for dev spec. If found, mark its presence ~~flag temporarily as 11h~~ then do init_done. If not found, determine entry not used & LDIR temp entry to there. If initializable, call its init routine. ~~Mark pres flag as 11h. Do init_done to~~ increment MAX_BANK. Do Write_BS_REGS. Do init_done.

~~Then eliminate and entries marked 1, reset presences to 1~~ Then do renum banks.

5/2/83

15h - device int priority (0-255)

assign bank #s by priorities.

RESSCT needs to be written

RESET needs to change (RESALL changes area in BLDSCT)

Addr of SYSCON changes to ~~5F4C~~ 5EEA.

~~After~~ At end of BLDSCT, call a routine to REDO_BANK_NUMS
examine device int priorities
renumber banks accordingly

Can a bank # be changed by doing
WRITE_BS_REG (BNA, x)
WRITE_BS_REB (ABN, y)
Taking care to never have y=any existing # other than x
Yes

~~Or do you have to do~~
~~write reset~~
~~WRITE_BS_REG (ABN, x)~~
~~write init_done~~

Could initially start with MAX_BANK > 7, thus ensuring no conflict on reassign.

On reassign,
reset MAX_BANK to 0

determine highest priority dev and make it bank 1
then determine 2nd highest and make it bank 2
etc
RAM gets highest bank #s.

RESSCT

First, go thru existing banks to determine if still there. If not, zero out present entry, then do as 2 pages ago.

Bank switching chip - ports are memory mapped.

RAM banks are a problem for warm reset.
Don't want to go ahead and do RAM test since this could destroy data in RAM.
Could read overhead bytes, then write a byte, read it back somewhere else.
If it changed, it's RAM, but don't know if it existed before.
Upon initial determination that it's RAM, write some unique ID to address 0000h.
~~to store bank # at OPEN address in SYSCON (bank # + 2)~~

Then at RESSCT
Read overhead bytes
Write a byte of 0 of device (diff from what was read)
Read it back to somewhere else
If same as written
then RAM
else if byte 0 > BS_MAX_BANK then new RAM (call NEWRAM)
search SYSCON for RAM w bank # in 0 and give it new bank #
else as written, search for device spec

implies that renumber_banks writes bank # to 0 of all RAM devices

Maybe now don't need to have BIT 7 set of bank # when first assigned. No, could still have conflicts. Or be smart enough to get rid of it before doing CALL_BANK etc.

DOES RESET on BEU wipe out existing BNAs? If so, we're in good shape and would not have conflicts.

To rename devices no longer attached:

check for end of table
look at bank #s
if BIT 7 is not set then device no longer exists else go on to next record

determine if this device has an open stream. If so, "close" it, ie zero out the STRMS data entry, mark presence flag as 0

Could just leave it up to the user to close the stream before removing the device.

CHANS changes to a close # on a stream attached to a record that no longer exist will not attempt to call a close routine but will still reset stream data

For NEWDEV:

search SYSCON for pres flag = 0 or pres flag = 80h

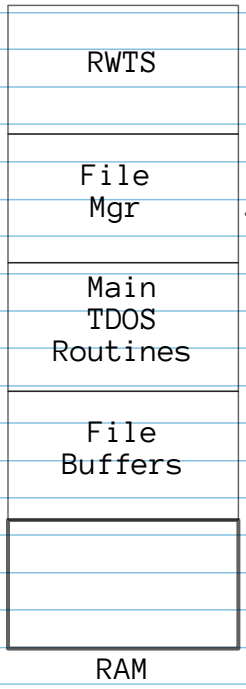
Change to EDIT

SRHSC - NXTONE

1st LD DE to LD DE, 18h
2nd LD DE to LD DE, 16h

TDOS ARCH

ROM/RAM layout



Read/Write Track/Sector - Benjamin stuff
Low level drivers

Performs functions ie OPEN, CAT, etc.
TDOS

interface is TDOS and set.

Interface to BASIC, syntax checking, etc. TDOS
init routines, file buffer mgr.

TDOS init routine moves TDOS code into RAM.

OPEN cmd handler allocates necessary buffer space.
CLOSE cmd handler deallocates said buffer space.

TDOS Cmds

- Open_File - Sequential mode, Random Mode. Opened sequential if no record length is provided.
- Close_File
- Set_Position
- Read_Record
- Write_Record
- Read_File
- Write_File
- Read_Byte
- Write_Byte
- Delete_File
- Create_File (or done by Open)
- Format_Volume or Create_Vol
- Catalog or Directory
- Rename

File Types

- Data - sequential, incl. DATA, SCREEN$, random (records)
- BASIC
- CODE

Access Modes

Sequential or random.

- Read Only
- Read/Write
- Append

Mode ::= Seq|Rnd

Protection ::= Read (only), Append, Write (Modify)

Basic cmds handled by disk cmd handler

- CAT
- FORMAT
- SAVE *
- LOAD *
- VERIFY *
- MERGE *
- ERASE
- MOVE

Basic cmds handled by OPEN routines
OPEN #

Basic cmds handled by CLOSE routines
CLOSE #

Basic cmds handled by INPUT routines
INPUT #

Basic cmds handled by OUTPUT routines
PRINT #

Stream # has to be passed so action can be performed on proper file.

NOTE: Most, if not all of the AROS init routine can go into the INIT module, thus freeing up 7Dh bytes in the home ROM.

MISC NOTE - VCR interface - future thoughts 7-13-83

It would be nice to have an interface to a VCR such that you could program the machine via the computer, inputting the name of the program as well. When a tap is put in enter a tape ID and starting #.

The idea is to have an on demand automatic listing of what is on a given tape and at what location. May also have write protect on certain portions of tape. If attempt to record on WPed portion, the tape will auto forward to an unprotected area.

Could display warning if not enough tape left to record set program.

Document ~~531N~~545N/9545N

Should state purpose of each test and test specifically for only those things related to the purpose (so anything marked"same" does not have to be explicitly tested for).

5.0 Test Spec

Exp. bank configurations
Test of initialization
Breakpt after BLDSCT to check SYSCON table

~~1 - no BEU no banks~~
~~2 - BEU attached with no banks~~
3 - BEU with 1 bank, no init routines, no AROS/LROS, non-storage device, boot pri = Aah, init pri = 30h
~~Check that RENUM bank works & that MAX_BANK & BS_MAX_BANK = 1~~

Verify that:

1. ASSIGN returns with carry flag set
2. MAX_BANK & BS_MAX_BANK = 1
3. Bank # entry at 5EF7 get 129 (81h)
4. XFER of overhead bytes is correct
5. First byte should be different so CHIR is called
6. Pres flag (5EF6) gets 1
7. Determine there is no init routine
8. Returns with HL = 5F0E
9. ASSIGN returns with carry flag reset
10. MAX_BANK and BS_MAX_BANK = 1
11. 5F0E gets 80h (call RENUM_BANKS)
12. Entry at 5EF7 is found with bit 7 set
13. 5FE9 gets F7; 5FEA gets 5E; 5FEB gets 30
14. End of table is formed
15. 5FE7 gets 1
16. ASSIGN returns with carry flag set
17. MAX_BANK and BS_MAX_BANK = 1
18. End of table is found (ASSIGN returns with carry flag reset)
19. EXINIT should call BOOTDEV
20. Determine device is bootable, but not higher than TS2000
21. Control passed to LED18
22. ~~Disk-like cmds are rejected~~

3 - 1 bank, bootable non-storage, boot pri = Aah, int pri =30h, no LROS/AROS
4 - 1 bank, initializable non-storage, int pri = 40h, no LROS/AROS
5 - 1 bank, bootable non-storage, boot pri = 50h, int pri = 20h, no LROS/AROS

Device Type	Boot Pri	Int Pri
A - boot, non-stor	AAh	30h
B - init, non-stor		40h
C - boot, non-stor	50h	20h
D - boot, stor	80h	30h
~~E~~	~~AAh~~	~~30h~~
~~F~~		~~80h~~
G - RAM		
H - RAM		
I - LROS - boot I don't think this has to be explicitly tested - it's covered by C	0h	0h

D can do a GOTO_BANK (LED18) to test disk-like cmds.

Make all tests without ROS since ROS always gets control except for machine lang AROS not-autostart which calls BOOTDEV.

z,u 3 - A, no ROS
w,u ~~4 - B, no ROS~~
y,u 5 - C, no ROS
x,u 6 - D, no ROS
~~7 - E, no ROS~~
~~8 - F, no ROS~~
s,z,w,u 7 9 - A,B
~~z,y,s,u 8 10 - A,C~~
~~z,t,u 9 11 - A,E~~
~~12 - B,F~~
z,w,y,t,s,v,u,x 8 13 - A,B,C,D
9 14 - G
10 15 - H
11 16 - A,G

Conditions to test

z	boot pri > 80h	A	
y	boot pri < 80h	C	
x	boot pri = 80h	D	
w	initializable	B	
v	storage	D	
u	non-storage	A	
t	2 with same int pri	A,D	
s	2 with diff int pri (A<B)	A,B	just plug into diff slots
	2 with diff int pri (B>A)	B,A	

4 Same as 3 for 1-6
7 does CALL_BANK to init routines
8-19 same
20 determine dev is not bootable
21 pass control to LED1

5 1-19 same
20 pass control to device

6 1-19 same
20 pass control to LED18

7 1-8 same
9 ASSIGN returns with carry flag set
10 MAX_BANK & BX_MAX_BANK = 2
11 Bank # entry (5F0F) gets 82h
12 XFER of overhead bytes is correct
13 First byte should be different, so CHIR is called
14 Pres flag (5F0F) gets 1
15 Does CALL_BANK to init routine
16 Returns with HL=5F26
17 ASSIGN returns with carry flag reset
18 MAX_BANK and BS_MAX_BANK = 2
19 5F26 gets 80h
20 (call RENUM_BANKS); entry at 5EF7 is found with bit 7 set
21 5FE9 gets F7; 5FEA gets 5E; 5FEB gets 30
22 Entry at 5F0E is found with bit 7 set
23 5EF7 gets 1
24 Entry at 5F0F is found with bit 7 set
25 5FE9 gets 0F; 5FEA gets 5F; 5FEB gets 40
26 End of table is formed
27 5F0F gets 2
28 ASSIGN returns with carry flag set
29 MAX_BANK and BS_MAX_BANK = 1
30 ASSIGN returns with carry flag set
31 MAX_BANK and BS_MAX_BANK = 2
32-35 Same as 18-21 in config 3

~~8 1-14 same as 1-14 config 7~~
~~15 determine device not initable~~
~~16-22 same as 16-22 config 7~~
~~23 5FE9 gets 0F; 5FEA gets 5F; 5FEB gets 20~~
~~24 5F0E gets 1~~
~~25 Entry at 5EF7 found with bit 7 set~~
~~26 5FE9 gets F7; 5FEA gets 5E; 5FEB gets 30~~
~~27 5FE7 gets 2~~
~~28-37 same as 26-35 config 7~~

~~9~~

8 A,B,C,D Test for 2 banks with same int pri and simply having 4 banks
1 Break at RENUM_BANKS and check SYSCON table
2 MAX_BANK and BS_MAX_BANK = 4
~~3 5EF7 found with bit 7 set~~
~~4 5FE9 gets F7; 5FEA gets 5E; 5FEB gets 30~~
~~5 5F0F found with bit 7 set - lower priority~~
~~6 5F27 foound with bit 7 set~~
~~7~~
3 break after RENUM_BANKS. Should be
5EF7 = 2
5F0F = 4
5F27 = 1
5F3F =3
4 MAX_BANK and BS_MAX_BANK = 4
5 Device C should boot up

9 G Test a RAM bank
1 (BLDSCT) first byte XFER'd should be same
~~2 next XFER back should be same~~
1 NEWRAM should be called
2 5EF6 gets 2 (82h?)
3 int fielder goes to RAM banks at 38h
4 upon ret, chunk spec (5EF8) should be right
5 MAX_BANK and BS_MAX_BANK = 1

10 H Test another RAM bank
1 after BLDSCT, verify chunk spec

11 G,A verify RENUM works with RAM
1 RENUM

What's in Exp. Banks

OPEN - ? - pops stream # and params off stack
CLOSE - ? - pops stream # off stack
SELECT - ? - pops stream # off stack
INPUT - ? - pops stream # and params off stack
OUTPUT - ? - pops cmd token and params off stack
Int handler - ?
Init routine - probably just a RET
Reset routine - probably just a RET
Boot adds - (Init = Boot)

All exp. bank EAPROMS can be the same except for overhead bytes.

```
0000       Dev spec
0001  00   Open address
0002  01   Open address
0003  00   Close address
0004  01   Close address
0005  00   Select address
0006  01   Select address
0007  00   Input address
0008  01   Input address
0009  00   Output address
000A  05   Output address
000B  00   Disk cmd handler address
000C  06   Disk cmd handler address
000D  38   Int handler address
000E  00   Int handler address
000F  00   Init address
0010  08   Init address
0011  00   Reset address
0012  09   Reset address
0013       Dev type:  0 = boot, non stor; 1 = init,
                      non-stor; 2 = boot, stor; 3 = init, stor
0014       Boot pri
0015       Init pri
```

300, 400, 500, 600, 100

```
     POP BC     ; stream # in C, 0 in B
LI:  POP BC     ; pop bytes off stack
     LD A,B
     CP EOD     ; until end of data
     RET Z      ; is found
     LD A,C
     CP EOD
     JR NR,LI
     RET
```

Except for 600 the 1st POP BC pops command token into B.

Stream commands

OPEN #
CLOSE #
PRINT #
INPUT #
RESET - cold start new devices
RESET * - cold start all devices
RESET # - warm start specified stream's device

Disk-like commands

CAT
FORMAT
SAVE *
LOAD *
ERASE
MOVE
VERIFY *
MERGE *

Stream Cmds - with Exp Bank D attached (default)

Test OPEN routines and test SRCHSC looking for 15 devices

2 OPEN #c,"m"
1. does not find match in SPEC_T
2. returns from SRCHSC with carry flag set, HL=5EF7h; DE=0Ch
3. PASSEM doesn't put anything on stack
4. Stream # passed on stack when device's OPEN routine is called
5. Before returning, OPEN writes stream data to STRMS with bit 7 set.

3 OPEN #c,"m" - with banks A&D attached
Test SRCHSC looking for 2nd device
1. returns from SRCHSC with carry set; HL=5F0Fh; DE=24h

1 OPEN #c,"m" - with no banks
Test SRCHSC when won't be found
1. ERRJ - INVALID DEVICE CODE appears on screen

4 OPEN #c,"m","string" - test passing one parameter (odd #) and test in string to ensure it doesn't take it as end of line
1. PASSEM should put "gn:rts" on stack

5 OPEN #c,"m","string1","string2" - test more than one parameter (even #)
1. PASSEM

~~6 OPEN #c,"m","s1:s2"~~

6 OPEN #c,"m"n - Test passing over floating pt #
Where n is a number whose floating point representation includes 3A, 0D, 22.

7 CLOSE #c
1. CLCHAN determines sream is attached to intelligent device
2. CALL_BANK to device's CLOSE routines with stream # passed on stack
3. Zeroes out stream entry

8 PRINT #c,a
0. On call to SELECT, determines device is intelligent, device's bank # goes into CURCBN, device's SELECT routine is called, stream # is passed
1. PASSEM passes a
2. Device's PRINT routine is called

9 INPUT #c,a
1. PASSEM passes a
2. device's INPUT routine is called

10-14 are init like; 15 is stream cmd

10 RESET - Test RESSCT - cold start all new devices with one device attached and not changed
1. RESSCT is called
2. ASSIGN returns with carry flag set
3. Overhead bytes are XFER'd read into 5FE9h
4. NOTRAM is jumpted to Bank is determined to not be RAM (jump to NOTRAM)
5. Device spec at 5FEBh should be same as at 5EF8h (so STEPON is jumped to and then GOTOIT)
6. ASSIGN returns with carry flag reset so NOMORE is jumped to
7. Check SYSCON table after return from RENUM_BANKS (5EF7h = 1)

11 RESET - start with device D and add device A (test adding of a device)
1. Second ASSIGN returns with carry flag set
2. End of SYSCON table marker moves to 5F26h
3. NOTRAM is jumped to
4. Device specs are different, so overhead bytes are moved down to 5F10h
6. 5F0Fh gets 82h
7. ASSIGN returns with carry flag set
8. Check SYSCON table after return from RENUM_BANKS

12 RESET - A replaced D (test replacing a device)
1. Check SYSCON table at return of RESSCT

13 RESET - D is remove (test removal of device without replacement)
1. At return from RESSCT, 5EF6h = 80h.

14 RESET * - have a strm open when done (test cold start of all devices)
1. CH_ADD pointer points past asterisk
2. RESALL is called
3. BLDSCT is called
4. Check that STRMS (stream data) has been zeroed our for streams 4-15 and std streams for 0-3
5. Upon ret check SYSCON table

15 RESET #c - test warm start
1. CH_ADD points past #
2. A gets stream #
3. DE gets stream data (8Ch)
4. Device's RESET warm start routine is called

Disk-like cmds - with device D

1 CAT "m"
1. CAT is called
2. Device spec is found in SRCHSC
3. PASSEM passes "A"
3. Cmd token is passed (0CFh)
4. Device's disk cmd handler is called

2 FORMAT "m"
1. FORMAT is called
2. Device's disk cmd handler is called, passing 0D0h (cmd token)

3 MOVE "m","old","new"
1. MOVE is called
2. Device's disk cmd handler is called, passing 0D1h and params

4 ERASE "m","f"
1. ERASE is called
2. Device's disk cmd handler is called, passing 0D2h and "f"

5 LOAD *"m","f"
1. In NEWDEV of SYNTWO, asterisk is detected
2. SRCHSC returns successfully
3. PASSEM passes "f"
4. LOAD token gets passed (0EFh)
5. Device's disk cmd handler is called

6 SAVE *"m","f"
1. Device's disk cmd handler is called, passing 0F8h and "f"

7 VERIFY *"m","f"
1. Device's disk cmd handler is called, passing 0D6h and "f"

8 MERGE * "m","f"
1. Device's disk cmd handler is called, passing 0D7h and "f"

8-17

For VAX, write tool to make cartridges.

Load tape emulating 2000
Download to VAX
Tool will ask questions as to where chunks will go (maybe), will figure out (most) overhead bytes and arrange 8K chunks properly to be burned.
Process is even simpler for 48K download (Spectrum or 2000)

32K ROM for system software

32 in Ext

- cannot have DOCK and EXT selected at same time, thus inhibiting the BASIC AROS search routines
- do not have direct access to system variables from chunk 2 of ext.
- chunk 3 of ext could not do bank switching unless bank 7 becomes the permanent permanently enabled chunk but function dispatcher stays in same place in RAM (6200)

16 + 16

- requires more bank switching in system software, but not much more than now. And less than 32 in ext due to system variables.

Microdrive

Auto-run facility that requires just a RUN cmd immediately after NEW or power-up.

FORMAT takes a cartridge name.

OPEN assumes mode - if file not found, it's open for writing else for reading.

4 files types

PRINT (data)	not PRINT (not data)
PRINT #	DATA (array)
LIST #	BASIC
CAT #?	CODE (bytes)

LIST # - outputs to a stream
INKEY$ # - inputs from a stream - one that's created by LIST # and PRINT #
CAT - displays only file name; returns amount of free space on cartridge
CAT # - sends output to specified screen

Protection mechanism is if a file name begins with CHR$0, the file name will not appear as a result of a CAT

CAT uses vol spec rather than device spec

Vol spec is a number rather than a letter

MOVE - can work with streams
- Syntax MOVE x TO y
- where x and y have the form #n | "m";n;"na"
- Works only with data files (PRINT #, LIST #)

How CALBAS works

Called by:
RST CALBAS
DEFW addr

CALBAS:
HL gets saved in (DR+H_L)
DE gets saved on stack
addr gets saved in (DR+ZILCH)
HL gets incremented to pt. past the DEFW
then gets exchanged with top of stack, which goes back to DE
Then 0, 8, addr is pushed
Then RETBAS does return to home ROM at addr
The called routine does a return to 8 which
brings us back to shadow ROM
The ERR_IN routine checks top of stack.
If 0, it loads HL with (DR+H_L) and returns to it

MAP is 32 bytes (corresponding to 256 sectors; nominal content 210 sectors, upper limit 230)

There is one MAP for each microdrive containing an open file.

The CHANS area contains 595 bytes for every open file the INPUT and OUTPUT address are 8 so the error RST is called and the shadow ROM takes over.

FORMAT finds all the good sectors and writes a header block on each one. A sector is 568 bytes including header preamble, header, data block preamble, data block.

Error Handler -> Parser -> ACTs -> Level 0 -> BEU -> Microdrive interface -> M1 -> Mc -> M8

To productize microdrives

- Change location of system variable.
  Less 8 bytes for network, all new vars would go up to 5CFD

CHANS starts at 6840h - m builds buffer in CHANS for each open file (done in Exp RAM)

Seems to return to DELREC (19E8) or (16C5) RESET (in EDIT) (reset calculator stack)

OPEN - if device "m"
  then sequential only (could make random access).
    I - write from start of file
    R - read
    A - write from end of file

Chunk 0 contains 8K ROM

~~chunk 1 contains 8K(?) RAM~~ probably use home RAM. Not enough room in chunk 2 for buffer - only 493 bytes unless change to allow max of 9 exp banks

Accesses system vars in chunk 2 and bacnk switching in chunk 3 or 7

Parameters to m are passed on the machine stack. m BASIC interface pops parameters off the stack, then use existing parsing routines.

CALBAS is replaced with CALL_BANK rather than JP RETBAS, can just do RET

A microdrive INIT routine could build room in home RAM for one output buffer. But where? After CHANS table - after MLV - moving everyhting up from PROG to STKEND.

If we have extra RAM in another chunk, we could possibly use that.

Somehow, m must know where the buffer is for each file. Stream # identifies entry in SYSCON table.

Could have another table mapping stream # to buffer addr. This table would be 32 bytes and be indexed by stream #. If m has its own RAM this table could go there, otherwise in chunk 2. ~~Actually, this table should also contain the addr of the MAP, thereby making it 64 bytes.~~

What does random access mean?

There is already a record # in header and flag indicating last record.

Records currently are placed in first available sector found from current location.

For RA, search for filename, compare record #.

If not match, check if last record.

MAX RAM Requirements - 9616 bytes 9.39K

8 MAPs - 256 bytes
Buffer add table - 32 bytes
16 Buffers - (each 583 bytes) - 9328 bytes
- CHBYTE - 2
- CHREC - 1
- CHNAME - 10
- CHDRIV - 1
- CHMAP - 2
- Header block
  - HDPREAMBLE - 12
  - HDFLAG - 1
  - HDNUMB - 1
  - ? - 2
  - HDNAME - 10
  - HDCHK - 1
- Read and written data block
  - DBPREAMBLE - 12
  - RECFLG - 1
  - RECNUM - 1
  - RECLEN - 2
  - RECNAME - 10
  - DESCHK - 1
  - CHDATA - 512
  - DCHK - 1

If we limit # of streams for m to 13, need only 7867 bytes = 8K, so could go with 8K ROM and 8K RAM

Could possibly use 8K in m exp bank for the actual DATA buffer and use 1K + 432 bytes in home RAM for other buffer info. Extra 32 is to contain addr of data buffer, unless '3' can be used for that. Maybe need to have buffer info addr in table, adding 32 bytes for this implementation.

9-2 CP/M - Microdrive

Cold start routine could look for CP/M cold start file and load it and run it. OR the CP/M cold start could be in ROM and executed as part of m cold start if CP/M present.

OR we could do like RUN and if the user types in CP/M -- or maybe RUN CP/M -- then the CP/M cold start is run.

How does BDOS-BIOS interface relate to ACTs.

Need to write BIOS to interface between BDOS and ACTs.

Current ACTs are for:

- CAT
- CLOSE
- ERASE
- FORMAT
- LOAD
- MERGE
- MOVE
- OPEN
- SAVE
- VERIFY

PRINT (and INPUT) work on buffer, but what happens when the buffer is full (or empty)?

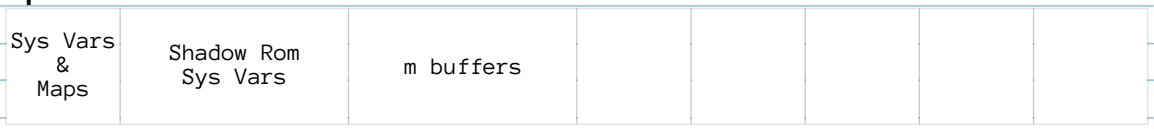
WRCD and RDCH do CALL 8 with 15FE on top of stack. ERR_IN detects this and gets proper addr out of channel. This addr is in the shadow ROM. These routines handle the buffer.

~~If CP/M sector on tape~~

FORMAT must search for CP/M boot sector and not erase that if found. Do the FORMATing around CP/M.

TS 2000 machine language AROS

Spectrum



TS 2000 without machine language AROS

VARS

So on 2000, MBUFF and MMAPS get initialized to PROG.

Or MBUFF and MMAPS go in new 32K RAM bank.

For m on current ROM, if don't put stuff into CHANS, when OPEN is done and CHANS is searched, 'm' will not be found (not in SYSCON) so RST 8 will be done. Shadow ROM will then be invoked. MBUFF and MMAPS can still be put before PROG. Buffer of course includes buffer vars.

32K RAM will be in interface unit.

If CP/M can take up lower 24K, can use 8K at 6000h for buffers and maps and new system vars.
Where does ROM reside?
It should go at 0 so RST 8 mechanism will work.
But CP/M expects RAM at 0.
Maybe RAM is in different bank than ROM?
ROM listens for addr 8.
RAM accessed by an OUT to a port
So, can RAM be accessed by ROM?

CP/M

BOOT and WBOOT load the CCP into the TPA in bank 1 at 0100h.
BOOT can load CCP.COM file from disk and place a copy in an alternate bank.
WBOOT can then load from alternate bank, thus avoiding all disk accesses during warm starts.

~~Drive Table consists of 16 words containing the addrs of the Disk Parameter Headers for each logical drive name A to P. ie supports 16 drives.~~

BOOT sets up the jump table at 0
loads the CCP
jumps to CCP

WBOOT sets up jump table at 0
reloads CCP
jumps to CCP

May have to determine what there is for memory. If CP/M is only available on 80K machine then always put jump table in 32K RAM bank. Otherwise, will have to check if 32K RAM is there, or if there is a RAM expansion bank, or whatever.

initialize BOOT_STACK at D2 up to 92 (64 bytes) - WBOOT only

CONOUT: fetch conout bit vector and jump to OUTSCAN. OUTSCAN scans all devices to see if they are output devices and if so see if they have a character to output and print it if device selected. CONOUT sends character to all selected output devices.

Basically what this wants to do is get the output addr of channel 'S' and output the character in C i.e. invoke SENDTV after a LD A,C or whatever is needed.

CONIN: Basically invoke INCH. Input character returned in A. CONIN returns character from first ready console input device.

~~Console input and output is only one device, but can be redirected by System Control Block.~~

Maybe disk and microdrives can be bootable and if they contain CP/M it'll take over. If not, it can lower its priority in the SYSCON table and go back to the BOOTDEV routine.

Microdrive initialization under CP/M may have to be radically different.

AUXOUT: output character in C to all selected devices.

There are bit vectors in the System Control Block (SCB) for CONIN, CONOUT, AUXIN, AUXOUT, LIST. CP/M supports devices numbered 0–15. A set bit in the vector indicates that device is selected. Supports 5 logical devices.

The SCB is located in the BDOS.

All character I/O operation are in 8-bit ASCII, upper/lower case with no parity. CTRL-Z (1Ah) denotes EOF for an input device.

Each of the 5 logical devices can be directed to any combination of up to 12 physical devices.

Bit 15 corresponds to device 0
Bit 4 corresponds to device 11
Bits 0–3 are reserved for future use.

An output character is sent to all selected devices.

An input character is read from the first ready device.

CONST: Returns in A
    0FFh if a console character is ready to read
    000h if no console characters are ready

LIST: Send character in C to all selected listing devices.

AUXIN: Reads next char from device into A.

```
@CTBL -

CHRTBL is the BIOS data structure describing physical I/O dev.
Contains:                          eg
      6-byte name                  'CRTwww'
      1-byte mode                  mb_inout_+serial+softbaud
      1-byte baud                  baud_9600

This table maps logical device # to physical device

mb$input          EQU   00000001B
mb$output         EQU   00000010B
mb$in$out         EQU   00000011B
mb$soft$baud      EQU   00000100B
mb$xon$xoff       EQU   00010000B

CP/M sectors are 128 bytes
Disk sectors are 256 bytes
Microdrive sectors are 512 bytes
```

The calls are probably CALL_BANKS with chunks 0,1,2 selected

Assumes interrupts handled by 2000 int handler and that streams 1,2,3 have original meaning even if not original channels.

Calls could be CALBAS's

```
CONOUT:     PUSH BC                 ;chunk 2 of Home must be selected
            LD A,2                  ;selects stream 2 - initially the screen
            CALL SELECT             ;  could now be 40/80 col screen driver
            POP BC
            LD A,C
            CALL SENDCH
            RET

CONIN:                              ;for current implementation
            LD A,1                  ;select stream 1 - initially channel K
            CALL SELECT
            CALL RDCH               ;wait for char, returned in A
            AND 7Fh
            RET

LIST:                               ;if 80-col printer does an OPEN #3,'80 col'
            PUSH BC                 ; during initialization
            LD A,3                  ;select stream 3 - initially channel P
            CALL SELECT
            POP BC
            LD A,C
            CALL SENDCH
            RET

CONST:      BIT KEYHIT,(IY-Y+FLAGS)     ;KEYHIT=5
            JR Z,NO_KEY
            LD A,0FFh               ;char ready to read
            RET
NO_KEY:     XOR A
            RET
```

```
CONST:      Compare input & output pointers of TAB
            If same then XOR A
                  else LD A, 0FFh
            RET

CONIN:      CALL CONST
            AND A
            JR Z, CONIN             ; wait for char
            Read char into A

CONOUT:     LD A,C                  - 40/80 col display? Search for channel S
                                    expects some variables.

LIST:       LD A,C                  - 80 col printer? Search for 80 col; if not
            JP SENDTV               present use P?

PUNCH:      LD A,C                  ; char in reg C is sent to dev
            JP W_TAPE               ; writes DE (make DE=1) bytes from RAM to
                                    ; tape. Not real interface for PUNCH
                                    ; No good since it will then write header
                                    ; each byte

READER:     JP R_TAPE               ; char from dev is returned in A (EOF is
                                    ; 1Ah

PUNCH:      JP LWT3
            LD IX,RAMLOC
            LD (IX),C
            LD C, 0Eh
            LD DE,1
```

```
PUNCH:      ;write a byte in C to tape
            RL C
            LD L,C
            LD BC,3B0Eh      ; 3B is timing constant/OE is MIC off & yellow
            JP SA_8_BITS
            RL L
            LD A,1
            SCF
            EX AF,AF'
            LD A,C
            EXX
            LD L,A
SA_BIT_2:   LD A,C
            BIT 7,B
SA_BIT_1:   DJNZ $
            JR NC,SA_OUT                    If kept as a call, need to do
            LD B,42h                        LD A,1
            DJNZ $                          OUT (F4),A
SA_OUT:     OUT (FEh),A                     LD A,80h
            LD B,3Eh                        OUT (FF),A
            JR NZ, SA_BIT_2
            DEC B                           Requires 29 bytes to set up CALL
            XOR A                           to ext ROM without bank switching
            INC A                           code in RAM.
SA_8_BITS:  RL L
            JP NZ,SA_BIT_1
            EXX
            EX AF,AF'
            RET

READER:                 ;return next char in A
            EXX
            LD B,0B2h
            LD L,1
LD_8_BITS:  CALL RD_BIT      (22 bytes long)
            RET NC (JP NC,EOF)    ; returns if error (maybe should return
                                  ; 1Ah in A)
            LD A,0CBh
            CP 0Bh
            RL L
            LD B,0B0h
            JP NC, LD_8_BITS
            LD A,L
            AND 7Fh          ; strip high order bit
            EXX
            RET
EOF:        LD A,1Ah
            EXX
            RET
```

WBOOT: loads BIOS disk starting at track 0, sector 2 (sector 1 contains cold start loader)

COLD START LOADER loads BIOS similar to WBOOT

Q: What does Benjamin's boot routine do?

It needs to load the first sector - the cold start loader for CP/M. It apparently does.

9-26 Microdrive Controller & CP/M

There's a switch on the box to switch between TS mode and CP/M mode.

In TS mode, comes up normally and 8K ROM (mapped to 0) in box traps addr 8 and if aslo turned on by an OUT. Probably turned off by an out.

In CP/M mode, comes up (mapped to 0) in 1K boot ROM. This ROM determines if booting CP/M (m or d) or disk based TDOS and loads appropriate boot routines at the top of memory. This ROM is then deselected and lower 32K maps to RAM in the box, upper 32K to RAM in TS2000.

Maybe could go between 8K ROM and 32K RAM in box via the upper 32K RAM.

To write chars to screen, need to turn off 32K RAM in box, XFERing char via upper 32K RAM.

So, where does this leave streams, etc for CP/M and DOS?

9-30

Boot routines for m CP/M in RAM will do the equivalent of LOAD* "m";1;"CP/M" CODE if it can access the 8K ROM.

CP/M LIST routine can send stuff out over RS232 channel. (But still should work with 2040)

Network uses 276 byte channel area (255 buffer). This can probably go in the same place as the m buffers but the size given to INSERT and DELREC would be the same. Still have 32 byte buffer addr table (1 word for each stream)

The 8K ROM and 32K RAM should be selectable by writing different values to a port, e.g.

```
OUT (F3),1 enables 8K ROM and keeps 48K home RAM selected
OUT (F3),2 enables 32K RAM and keeps upper 32K home RAM selected
OUT (F3),0 enables 64K of home bank
OUT (F3),4 enables boot ROM and 48K home RAM
```

Coming up in CP/M or TDOS mode, no 2000 initialization is done. However we still want streams (in TDOS mode at least). TDOS boot routine could perform the necessary system initialization.

Strict CP/M environment probably doesn't have any 2000 initialization nor does it assume any system variables.

MBIOS will need to have routines to print char to screen and have int handler to read keys

at 38h could be a jump up into upper 32K that would select 2000 ROM to handle interrupt (CONIN)

Can also have 40 col screen driver up there that will put the bits into the display file (CONOUT)

Tape I/O same as written.

What about LIST routine? Can use SENDTV as in home ROM for 2040. What about RS232 printer? Could maybe determine that there is no 2040 and then send via RS232. The problem is that LIST outputs only one character and would therefore do this for each character.

What about network support in CP/M? Devices cannot be redirected in 2.2

10-3

As part of BOOT, could see if 2040 printer is attached. Apparently the routine to check this is at F0C in Logan:

```
IN A,(FB) (or test bit 6)
ADD A,A
RET M      ; ret if no printer
```

Could set a BIOS system variable indicating whether or not it's attached. The LIST routine then checks this and outputs to 2040 if present. This is easy but what about RS232. Do we output to that all the time or just when there is no 2040? Is there any way to check status of RS232?

Devices can be redirected via IOBYTE!!

9-29 M

The insertion routine to initialize the shadow ROM first makes room for the 39h bytes it uses for system vars by calling INSERT with HL=DRIVES-1. Then it LDIRs a table to DRIVES that fills in the first 19 bytes. This includes a CALBAS subroutine. Other 20 bytes get FF.

CALBAS in shadow ROM fills in the 2 holes in the routine in RAM (H_L and ZILCH - addr to be called). So when the ret is made to the home ROM, it is actually made to the RAM routine which is:

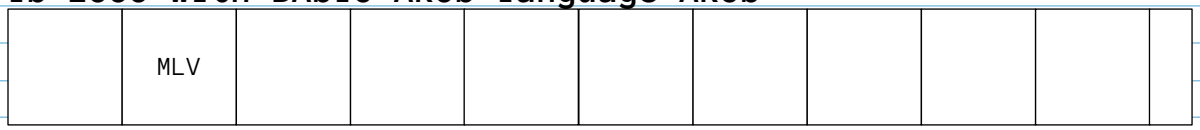
```
LD HL,____(H_L)       ; restores HL to what it was before RST CALBAS
CALL ____ (ZiLCH)
LD (H_L),HL           ; saves HL since g destroys it
RET (to 8)            ; this is then restored to HL in
                      ; the ERR_IN routine in shadow ROM
```

Thus, all registers are preserved.

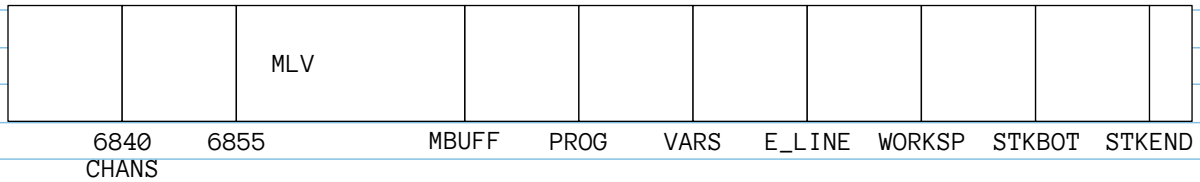
MMAPS and MBUFF cannot be inserted before PROG because ARSBUF can change size and INSERT would not handle them. So, they would have to go before ARSBUF and if they vary (at least the first one) the m code would have to perform the update on the other pointer.

May have to go with Spectrum implementation and put them in CHANS and suffer the consequences with machine language only AROS's in first new. A MLO AROS probably won't access m anyway.

TS 2000 with BASIC AROS language AROS



TS 2000 with machine language only AROS



The m INSER_ routines can check the SYSCON table to determine ~~if there is a machine language AROS. If so, do the INSERT at (PROG)-1.~~ If not,

If BASIC AROS, INSERT at (ARSBUF)-1
If MLO AROS, INSERT at (PROG)-1
Else, INSERT at (CHANS)-1

Note: ARSBUF never gets initialized if there is not a BASIC AROS.

We could put the entire CALBAS routines in RAM. This could do the necessary OUTs to enable/disable the shadow ROM while still retaining control. Question is, how much room would this take and could it fit into the sys vars area.

Actually, would only need to add the OUTs to the Spectrum CALBAS SBRT in RAM. This prevents the necessity of making sure the next instruction in home is the right one.

Sinclair does the INSERT for sys vars because DRIVES is at the end of sys vars and after initialization (CHANS_) is set to DRIVES, so room actually has to be made. The 2000 however has some room after the sys vars, so no room needs to be made. We can actually declare room for the maps right after the sys vars.

Our last var is at 5CCB
Additional m vars (39h) 5D04
Microdrive maps (max of 8) 5E04
SYSCON table starts at 5EEA
# of free bytes E6 (230 decimal)

Put together something for Mike's product specification of m detailing current knowledge and assumptions of m software and CP/M interaction.

To do random access, remove restriction that if file found it is open for read only. When looking to read a random record, check last record flag to determine existence. On write, do same and if last record found and rec num to be written doesn't yet exist, will have to remove the flag and put on new record.

Overwriting an existing record may require doing an ERASE function first.

ERASEing a random record that is the last record will require finding the new last record. This required once around the tape. Then around again to locate it.

Do we support random ERASE? NO?

Microdrive

TS 2068 Mode

All BASIC syntax and run-time errors produce a RST 8 instruction. The microdrive hardware traps address 8 and the lower 8K is then mapped to the microdrive ROM.

If a real error has occurred, processing is returned to the home ROM.

If the error is a microdrive command, the microdrive ROM performs syntax checking, returning to the home ROM if an error is found. If there is no error, the command is executed. Upon completion or detection of an error, control is returned to the home ROM.

Other than trapping on 8, control is transferred between ROMs by executing an OUT instruction on a <TBS> port. There will be a routine in RAM (chunk 2) that facilitates the microdrive ROM using routines in the home ROM by enabling the home ROM, calling the desired routine, and enabling the microdrive ROM.

The microdrive ROM requires access to the TS2068 system variables as well as several variables of its own to be placed in chunk 2. There will also be room reserved in chunk 2 for up to 8 microdrive maps, one for each drive. Also in RAM will be a buffer for each stream open to a microdrive file or to the network. The buffer for a file contains 512 bytes of data. The network buffer contains 255 bytes of data.

The Sinclair code to handle RS232 and the network probably only needs to be modified to account for the new location of the microdrive ROM system variables and the network buffer.

The code for the microdrive needs to be modified to handle the commands as defined in the the TMOS Functional Specification. The most significant addition is BASIC support for random access files.

CP/M Mode

The boot ROM invokes a routine in the microdrive ROM to load the CP/M boot file off of microdrive 1, into the top of home RAM. The boot ROM is then deselected "permanently". The boot routine continues loading CP/M from microdrive 1 and loads the jump table in RAM at address 0.

The CP/M microdrive BIOS uses code in the microdrive ROM to execute most of its commands. BIOS supports tape I/O at the byte level. It can also support RS232 output or the 2040 printer (by accessing home ROM).

Screen output is 40/80 column format. Chunk 2 of home ram needs to be

switched in to write to the display file.

So as to be able to use the microdrive ROM routines, the physical record size will be 512 bytes. However, to be compatible with CP/M, the BIOS will treat each physical record as for 128 byte logical records

IOBYTE at 0003h

7	6	5	4	3	2	1	0
LST		PUN		RDR		CON	

Values:

```
     CON:  0 - TTY:   RS 232 I/O
           1 - CRT:   kbd input/40 col output
           2 - BAT:   (RDR input/LST output)
           3 - UC1:   kbd input/80 col output
     RDR:  0 - TTY:   RS 232 I/O
           1 - PTR:   cassette input
           2 - UR1:
           3 - UR2:
     PUN:  0 - TTY:   RS 232 I/O
           1 - PTP:   cassette output
           2 - UP1:
           3 - UP2:
     LST:  0 - TTY:   RS 232 I/O
           1 - CRT:   kbd input/40 col output
           2 - LPT:   2040 printer
           3 - UL1:   Centronics printer
```

BOOT routine must set the values of IOBYTE to the defaults
CON:=CRT: RDR:=PTR: PUN:=PTP: LST:=LPT: if 2040 plugged in, else UL1

The routines using the logical devices must check IOBYTE to determine what physical device to use.

If we do go with CP/M on tape, save it off so that it doesn't append in a CAT.

We might have a problem with addr 38h. CP/M uses it to jump into DDT or SID when running in debug mode. TS 2068 uses it as kbd interrupt.

~~128 byte physical sectors would yield 1020 sectors per drive. Sector # would then be 2 bytes.~~

~~If we can reselect boot ROM, warm boot is a breeze. Still is if CP/M loaded from m.~~

10-10

CONOUT, CONIN, LIST, CONST, PUNCH, READER could possibly check IOBYTE and do its own version of SELECT - setting CURCHL to appropriate address - and then using appropriate ROM routines - SENDCH, RDCH, INCH, WRCH.

Can input/output a byte to RS232, but must first set baud rate var DR+BAUD. Then can use IN_BYQ (rets in A) for CONIN, READER, OUT_T2 (writes A) for CONOUT, LIST, PUNCH.

For CONST RS232, do

```
          LD HL, SER_FL+DR
          LD A,(HL)
          AND A
          RET Z           ; ret with A=0 if no char ready
          LD A,0FFh
          RET
NO_CHAR:  XOR A
          RET
```

```
System vars:    BAUD      (must be set before call)
                SER_FL    - 0 no need to set
                IOBORD    (must be set before call; seems to be 0 so no
                           need to set)
```

However, if there isn't one, there'll never be one using this scheme. Could modify IN_BYP code for CONST that will check port

For network

```
     Input:     IN_BYT    - looks in buffer for a char
                            takes it if one is found
                            else waits for further packet
     CONST - returns A=0 if buffer empty, else A=0FFh.
     IN_BYT assumes buffer area and its associated data is set up.
                CURCHL    set to base of channel area
                NCOBL     'output' length (should be 0 to avoid error)
                NCIBL     present length of buffer 0-255
                NCCUR     last char taken
                NCB       buffer
                NCTYPE    type of packet ('DATA' or 'CLOSE')
                NCIRIS    destination station (m or 0 for broadcast)
     IOBORD
                NCSELF    own station #
```

network OUTPUT - OUT_N
assumes buffer area set up.

```
          CURCHL     - base of channel area
          NCIBL      - input length (should be 0)
          NCOBL      - output length
     IOBORD
          NCIRIS     - dest station #
          NCNUMB     - block # (0-65536)
          NTSTAT     - own station #
```

May want to keep ~~some sys vars to indicate~~ another copy of IOBYTE that is compared to IOBYTE before calling routine for current device. If they are different, call an initialization routine for the newly selected device.

Fixes for 5.0

AROS init routine to INIT module

STREAM cmds should not have cmd token passed on stack but should have stream number passed.

in (EDIT) SELECT, addr 560, line 1060 - LD A,(CURCBN) is redundant

in EDIT (NEW), addr 7B and 7D - IM1 can go to EXINIT, NOP can be removed

in INIT, RESSCT (NOTEND) - remove CALL ASSIGN

in GOTOIT, LD DE,24 should be:

```
      INC HL      ; zero out existing device spec
      INC HL      ; to ensure new one is different
      LD C,0
      LD (HL),C
      LD DE,22
```

in INIT, don't check AROS lang type beyond BASIC, machine lang so it can be controlled by an exp bank LROS

in INIT, on RAM test, 2nd byte read back should be compared to (POLLEB) not (EXINIT)

in RESSCT, also bit 7 of bank # not set if RAM, after call to CHIR next SYSCON record is completely skipped. To fix: ~~change INC HL to DEC HL,~~

STEPON should set but 7 of bank #

STEPON should first do:

```
            DEC HL                  ; point to bank #
            LD A,(MAX_BANK)
            SET 7,A
            LD (HL),A               ; save new bank # w/ bit 7 set
```

then change LD DE,22 to LD DE,23.

after CALL CHIR, replace INC HL with:

```
            LD DE,24
            SBC HL,DE               ; point to bank #
```

also remove

```
            LD A,(MAX_BANK)
            SET 7,A
            LD (HL),A               ; before the CALL CHIR
```

in RDCH, check CURCBN - if not 0 don't set ECHOREQ

~~in EDIT, move the resetting of the ideo mode up to the INIT label (re: BUG LIST #11, 8/3/83). Currently just before (C) message printed~~
Carol took care of this

for INPUT# CHR$(1), the CHR$(1) has to be passed on stack to exp bank

in PRINT routines - when encountering a semi-colon (;), check to see if printing to an intelligent device. If so, do a WRCH, if not, don't. (P_SEP in SYNTWO)

Must handle CP option re in TDOS.
~~so once opened,~~ once a file created with CP is loaded or merged, it cannot be listed (LIST,LLIST) or saved (SAVE)

POLLEB & RESSCT

~~ASSIGN~~ - do not do XFER_BYTES while in INIT mode. Algorithm becomes (for POLLEB):

do ASSIGN for all banks
for each bank do the guts of POLLEB
call RENUM_BANKS

Changes in:

POLLEB, INDUN, NEWRAM, CINIT

WRITE_BS_REG - when writing to cmd reg, only want to send 4 bits, so after writing LSN nybble, don't want to write MSN. Fixed by checking to see if writing to addr C000. If no, don't write it.

in INIT - initializing machine code AROS offset 4D, put in:

```
     LD HL,DRIVES
     LD BC,21
     LD (CHANS),DE
     LDIR
```

maybe - in CONT, do:

```
     LD HL, MEMBOT          (LD HL,5C92)
     LD (MEM),HL            (LD (5C68),HL)
```

Note: if an LROS wants chunk 3 enabled, it must do so itself. Otherwise, in GOTO_BANK, the bank switching code is deselected after BANK_ENABLE and before the address is jumped to.

Maybe look into the possibility of preserving IX in calls to DISPATCH, CALL_BANK, GOTO_BANK. Done with replacing the JP (IX) with a RET with proper addr and returning addr on top of stack. Have to consider the game playing involved with the stack.

In building SYSCON table, if HL gets to be TEMPREC, STOP. This allows there to be more banks out there than will fit into SYSCON. They have been assigned bank #s, but the software will not support them since MAX_BANK will not be >10 on the second time around. Also ensures that SYSCON table will not overflow into the stack.

Byte 13h in home ROM is now the version # (FF for first release)

in INIT, at line 638, jump is to wrong place. Should just skip next instruction.

in EDIT, SEL_2, line 1054 - replace 6 INC HL's with

```
     LD DE,6
     ADD HL,DE
```

remove LD A,(CURCBN)

in AROS initialization, line 119 - save a byte by replacing

```
     LD HL,DRIVES
     DEC HL
```

Wtih LD HL,DRIVES-1

10-11-83

Control Keys using Symbol Shift as Control

	Key	TS hex value	key	TS value
Ctrl	A	E2	N	2C
	B	2A	O	3B
	C	3F	P	22
	D	CD	Q	C7
	E	C8	R	3C
	F	CC	S	C3
	G	CB	T	3E
	H	5E	U	C5
	I	AC	V	2F
	J	2D	W	C9
	K	2B	X	60
	L	3D	Y	C6
	M	2E	Z	3A

Using E mode for control

Key	TS value	ASCII Ctrl Key	Key	Value	ASCII Ctrl Key
A	E3	(1)	N	A6	(E)
B	C4	(2)	O	BE	(F)
C	E0	3	P	AD	10
D	E4	(4)	Q	B2	11
E	B4	5	R	BA	12
F	BC	(6)	S	E5	13
G	BD	(7)	T	A5	(14)
H	BB	8	U	C2	15
I	AF	9	V	E1	(16)
J	B0	(A)	W	B3	(17)
K	B1	(B)	X	B9	18
L	C0	(C)	Y	C1	(19)
M	A7	(D)	Z	B8	1A

```
LOAD* "m";1;"CPM.SYS" CODE
```

BOOT ROM = upload code into chunk 4

Q: What RAM is selected when boot ROM is selected?

This code does:

```
           LD BC,1
           LD (DR+DR_STR1),BC
           LD BC,7
           LD (DR+N_STR1),BC
           LD BC, NAME
           LD (DR+N_STR1+2),BC
           SET 4,(IY+FLAGS3-Y)
           LD A,3
           LD (DR+HD_00),A            ; type code (3)
           LD BC,LENGTH
           LD (DR+HD_0D),BC           ; store length
           LD BC,START
           LD (DR+HD_0D),BC           ; store start addr
           OUT (F3),MROM
           CALL GET_HD                ; creates channel and opens the file
           JP H_SORT                  ; but have to return and jump
NAME:      DEFB 'C'                   ; to START
           DEFB 'P'
           DEFB 'M'                   ; is there any reason why this has
           DEFB '.'                   ; to load a boot routine
           DEFB 'S'                   ; and not the real CP/M?
           DEFB 'Y'
           DEFB 'S'
```

Something has to load low RAM. Waht does? The BOOT routine in CP/M.

If the CP/M can change, the boot routine may change, so these should both be on m cartridge. So boot ROM should load boot routine.

** But see next page!

'CP/M BOOT' - loaded somewhere in RAM

10-13

bias = B000h
loadp = E400h - leaves 7k
ccp = E400h - base of CCP
bios = FA00h - base of BIOS
biosl = 0300h - length of BIOS (this goes up to FD00h - still 300h left)
boot = FA00h
size = 1900h - size of CP/M system
sects = 32h - # of sectors (128 byte) to load (38h if go up to FFFFh)

Don't really need this (sects). Load until last record encountered. In fact, all it really wants to do is the equivalent of a LOAD* "m";1;"CP/M" CODE ccp, size, and ccp, size can be taken as whatever is in file header. So 'CP/M BOOT' can be in the boot ROM and LDIR'd into RAM

So, just what does GET_HD do and what does it need set up before it's called?

D_STR1	2 byte drive #	1
S_STR1	stream #	irrelevant
L_STR1	dev spec	"m"
N_STR1	2 byte length of filename	7
N_STR1+2	2 byte addr of filename	TBS
HD_00	file type code	3
HD_0B	length of data	0
HD_0D	start addr of data	0
SC_00	file type code	3
SC_0B	length	0
SC_0D	start	0
FLAGS3	set load flag	10h (SET bit 4)

calls OPENF (VHSOFT) which calls CREATE (HISOFT)
[CREATE has to change for TMOS]
Create sets IX and creates a m channel
Call DRVSEL
Loads FF into (SECTOR)
Map all free sectors and look for file
Reads first rec if it exists

~~[* This will have to be from a var so can be 4x this for CP/M or CP/M needs its own GET_HD that calls CREATE, DRVSEL, OPF1 *]~~

~~For 128 byte sectors, need 128 bytes per map (32*4). MAPLN is constant in CREATE.~~

in LOGO, specifying device Z: gets cassette I/O

~~Anything calling RDBLK, needs to be rewritten for CP/M to RD1~~
~~So CP/M GET_HD consists of~~
~~CALL CREATE~~
~~CALL DRVSEL~~
~~CALL WTHDR~~
~~CALL RD1~~
~~CALL ECD1~~
~~CALL RD1~~
~~CALL ECD1~~

TMOS must support variable length records so just open files with [128]. Alternatively, use blocking/deblocking.

~~Note: It seems that if the constant lengths HD_LEN, D_LEN, HD_MAX, DCHK, WORDCOUNT for PUSHMP (VHSOFT) and POPMAP are accessed via sys vars, then the bulk of the code will work unchanged for CP/M. HD_LEN will probably be the same for both.~~

H_SORT (SH_SAVE) analyzes the header read in. If ok as a CODE file, picks up start and length and

```
CALL GET_BL (SH_SAVE2)
CALL GETMBL (does a divide by 512)
CALL READ_1           ; ends up putting code in RAM
JP SLVMEX
CALL CLOSIX
CALL DELIX  [needs to change for TMOS]
JP END1 - BUT DON'T WANT TO GO HERE
```

So, probably need separate CP/M copy of H_SORT that JP's to CP/M BOOT and hopefully could

```
CALL GET_BL           ; may have to have diff versions of
CALL GETMBL           ; GET_BL and GETMBL
CALL READ_1
CALL CLOSIX
CALL DELIX
JP BOOT
```

10-14

```
BOOT:       OUT (32KRAM),ON
            ;set IOBYTE
            IN A,(0FB)
            ADD A,A
            JR M,NO2040
            LD A,95h                ; CON:=CRT, RDR:=PTR, PUN:=PTP, LST:=LPT
            JR BOOT1                ; print signon msg
NO2040:     LD A,0D5h               ; CON:=CRT, RDR:=PTR, PUN:=PTP, LST:=UL1
BOOT1:      LD (IOBYTE),A
            JP GO_CPM
WBOOT:
            ; reload from m drive
```

GO_CPM must initialize micro drive sys vars and any used be routines called in home ROM.

default DMA buffer area is 80h.

Channel is in chunk 2 of home RAM

So, will have to LDIR 128 bytes up to BIOS memory, select 32K RAM, and LDIR it to the DMA address. Will have to do a similar thing for the FCB.

Go other way when writing.

Read and Write have to handle this.

Possible modification to these BDOS functions:
15 - Open File
16 - Close File
19 - Delete File
22 - Make File
23 - Rename File
17 - Search for First
18 - Search for Next
35 - Compute File Size

do sched for CP/M by Fri 10-21
listing all assumptions (VAX ready, BSO ready, etc) CP/M code

list all milestones (routines completion dates etc)

Assumptions

How works as follows:

Code running in upper 32K can select 32K RAM, lower 32K of home bank, and 8(+)K ROM (either directly through I/O port or RST 8)

Will use modified code in 8(+)K ROM as part of BIOS

List all milestones and the necessary assumptions for each.

Assumption

m software modified to handle logical records - thus, CP/M does not need to do blocking/deblocking.

BIOS software will use m routines as much as possible. Some routines have to change for 2068 impl. but except for handling logical records, routines should work when called on Spectrum except won't have 32K RAM. Perhaps, could use 2068 and controller box w RAM and Sinclair m ROM and Spectrum LROS - Probably no reason why BIOS routines couldn't (largely) be tested pretending CP/M uses 512 byte sectors.

RAM in IM1 and use 2068 kbd handler. But when doing a CP/M read of kbd use a translation table from 2068 codes to CP/M codes - most significantly control codes (probably result of E-mode)

CP/M Functional/Architecture/Design Spec

To Test:	Need:	
RDR: PTR: PUN: PTP:	2068 as is. could hand asm test prog could write out a a string stored in RAM then in BASIC do: DIM A$(10) FOR I=1 TO 10 LET A$(I)=USR read NEXT I PRINT A$	But as this handles a byte at a time, timing may be critical - maybe not - if necessary timing is done here since it does wait 3B DJNZs before writing byte
CONIN: CRT: UC1:	~~2068 as is~~ Actually need 32K RAM so can do translate TAB at interrupt time.	Simply translate Sinclair key code in LAST_K into full ASCII. Might want to implement TAB in CP/M int handling routine that takes what is in LAST_K if KEYHIT and puts it in at end of TAB.
CONIN: TTY RDR: TTY	I/F box with RS232 device sending data. Probably don't really need to test.	
CONOUT: TTY RUN: TTY LST: TTY	Same but data sent to device. Problem with all these is is initializing channels. Probably handled by STAT.	
RDR: UR1 PUN: UP1	If this even goes - just need 2 I/Fs talking to each other	
CONOUT: CRT LST: CRT	2068 as is. Just need Dave's 40-col driver (on disk)	
CONOUT: UC1	2068 as is. Need Dave's 80-col driver.	
LST: LPT	2068 as is. Uses current printer code unchanged.	
LST: UL1	Centronics driver software.	
CONIN CONOUT READER PUNCH LIST	On the higher level, need IOBYTE at 3 in 32K RAM, but could temporarily use some add in the 48K RAM.	

STAT utility probably does any necessary initialization and therefore the necessary code could be in STAT.

CONST:	For most, just 2068. For TTY and UR1, need Sinclair m ROM. For CRT, 32K RAM for TAB impl.	

Coldstart loader: Could test as LROS using 2068 wiht controller box and drives. Need new m software for this.
Could use Spectrum LROS that instead of downloading would execute the boot loader.

BOOT: 32K RAM
WBOOT: We could start out testing and whatnot as if we had only 32K of RAM, which would start at 8000h in home bank.

HOME: Just RETs
SETTRK:
SETSEC:
SECTRAN: (almost)

SELDSK: Spectrum with drive

SETDMA: Needs IOBYTE

READ: Needs new m software (could be done using Spectrum in part
WRITE: assuming 512 byte buffers.) Really needs 32K RAM too.

LISTST: Can implement for 2040 by checking buffer, but may have to just return 0 for others. Anyways, basically just need 2068 and/or controller box ROM.

FLUSH: new software (need more specifications)
DISCARD: 2068 - just discards buffer contents
MOVE: 2068 I guess.

STAT probably needs to be modified unless keep 2 copies of IOBYTE and perform initialization and deinitialization when they differ.

11/30 will get BDOS src and BiOS sample from DRI

Possible path

Use BSO software on VAX
Download to PROLOG
Have ROS download to RAM like cold start loader
Emulate on ZDS

1. At first do for non-disk stuff requiring only 2068
2. Then assume controller box is available with Sinclair ROM modified for 2068 addresses (new sysvars and ROM routine addreses).
3. Then assume m ROM modified for logical records.
4. Then assume 32K RAM available. With or without boot ROM since we can do that from LROS.
5. Everything

1. Cassette I/O 1/3 day
   2040 output 2/3 day
   40/80 column output 1 day
   HOME,SETTRK,SETSEC,SECTRAN,LISTST 1 day
   kbd input w/o TAB - translating controls read from LAST_K 1 day
   10/28
2. RS 232 I/O 3 days incl some channel initialization
   SELDSK 1 day
   11/4
3. READ/WRITE 2 weeks
   cold start loader (LROS ver) 1 week (CP/M on m)
   FLUSH,DISCARD,MOVE 1/2 week
   12/2
4. BOOT,WBOOT 1 week
   IOBYTE impl of CONIN,CONOUT,SETDMA,AUXIN,AUXOUT,LIST 1/2 week
   kbd input w TAB 1/2 week
   CP/M kbd hdw
   Here by Dec. 16 VACATION
5. BDOS changes? 2 weeks
   Any surprises 1 week

Total 8-9 weeks

10/25

2040 driver

Uses SENDTV (500h) - presumption made that P_POSN and PR_CC are set. P_POSN is col # in printer buffer (initially 0). PR_CC is addr of nect char for printer (initially 5B00).

Best accomplished by calling CLPR (A35h). Before calling, do SET PR, (IY-Y+FLAGS).

SENDTV will then put what is in A into the printer buffer.

40 col driver

Uses a different version of SENDTV.

What parts of home ROM can be used?	What's changed
LDTVCU	P_LET
~~SENDT~~	P_NL
STTVCU	PR_TV1
LSU5	PR_TVB
LSU7	PR_TV
MESTXT	LSV7
PR_TV2	SENDTV
PUTSYM	SENDT
LPV4	TV_CC
WR_TAB	SETTVC
ATTBYT	SET-AT
PUTMES	P_AT
	S_TV_1
	S_TV_2
	P_COL

Should be able to save about 500h bytes by using routines in home ROM.

Need to keep new char set in BIOS.

Assumptions made by SENDTV
- bit PR of (FLAGS) is reset
- (S_POSN)
- (DF_CC)
- bit LHS of (TVFLAG) is reset
- (P_FLAG)
- (CHARS)

CP/M

DRI has:
4K BIOS
4K BDOS

1. CP/M with no additional RAM
   get list of options ROMable/wafer - what implications
   0000-0100 address problem

   What percentage of apps will run in 32K

2. Personal CP/M BIOS & BDOS BIOS spec: see if can get this
   add functionality
      cursor control, etc

see if I can solidify my sched

What changes to BDOS and will I need to make any.

How necessary is the Disk Parameter Header and Block. Is it used by BDOS? Or only by BIOS? What implications of not having a directory here? Apparently we'll have one.

Call in with status report each day

Contacts
   Steve Williams - project manager of consumer OS's
   Steve Cavenden - mgr Personal CP/M

Bruce Arwine 617-229-6222

~~Thur~~ - Fri: 11:00am

Monterey - Hyatt Del Monte - 408-372-7171

Thu United 1147 1:40-2:14 SF-Mont.
she'll have car

Thu Bradley ~~4:25~~ 1:45 United 527 Amer 341
San Jose ~~9:16~~ 6:11
car in San Jose

Sat San Jose 9:21 American 596
Chi 3:15

Mon Chi 11:29 United 316
Bradley 2:20

Directions to Hyatt

San Jose - 101 South to 156 West to Highway 1 South in Monterey
look for rt. 68 but don't take it
next exit says DOWNTOWN MONTERREY
rt at end of ramp
left hand lane
left at light
Hilton
Mark Thomas Drive
upon hill by Shell station

Spectrum LROS
modified to contain
cold boot

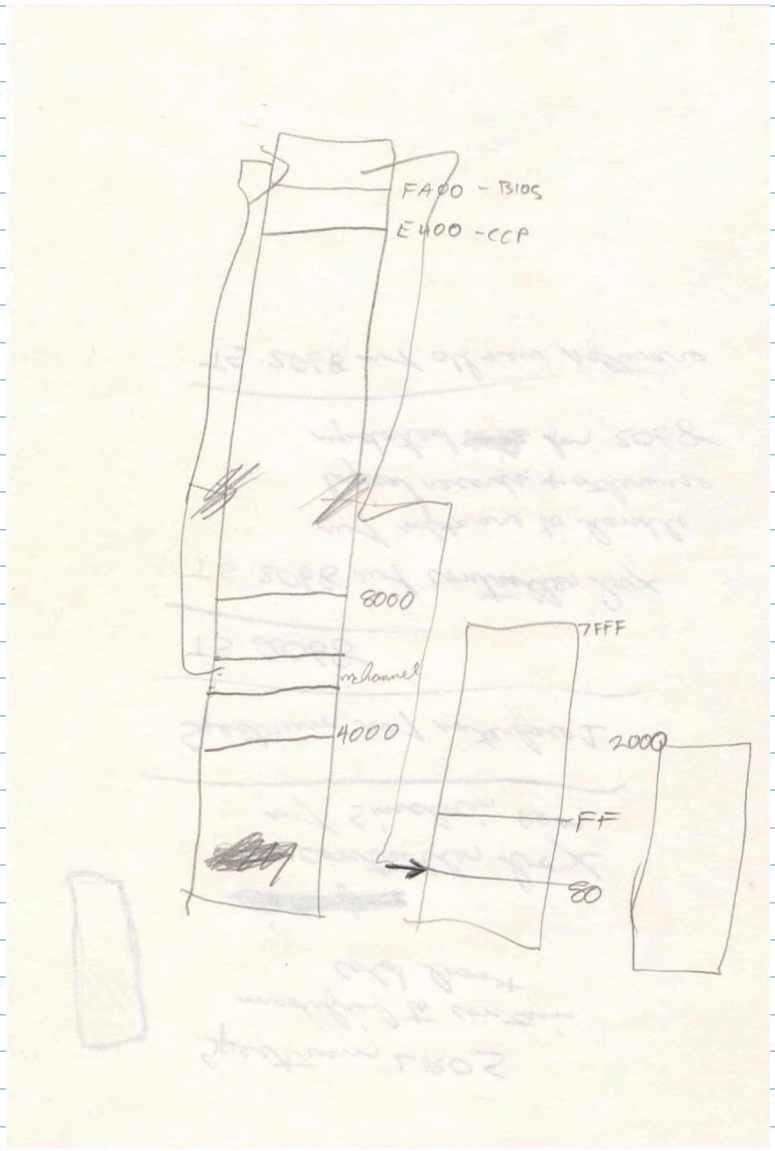
~~[illegible]~~ m controller box
w/ Sinclair ROM

Spectrum w/ interface 1

TS 2068

TS 2068 w/ controller box
w/ software to handle
logical records & otherwise
updated ~~to~~ for 2068

TS 2068 w/ all new software


FA00 - BIOS
E400 - CCP
8000
4000
7FFF
2000
FF
80

P. 2209 Rev.

** NOTE **

No ticket will be issued unless form is properly coded and has appropriate signature. All personal travel must be paid for prior to issuance of ticket.

Current Year (Jan.-Dec.)			
Number of trips to date	0	0	0
Location	Far East	Europe	Domestic

Reason: Visit Digital Research re CP/M.

TIMEX CORPORATION

TRAVEL ITINERARY FORM

NAME LON HILDRETH HOME PHONE 754-5822 BUSINESS PHONE 573-5298 DATE 10/31/83

FLIGHT SCHEDULE:

driver 11:45 - 12:00

DAY	DATE	AIRLINE	FLIGHT NO.	POINT OF DEPARTURE	DESTINATION	LEAVE	ARRIVE	CLASS	STA
Thurs	11-3-83	UNITED	~~527~~ 433	BRADLEY	CHICAGO	~~1:45 pm~~ 4:25	3:00 pm 5:44	econ	
Thurs	11-3-83	~~AMERICAN~~ UNITED	~~341~~ 583	CHICAGO	SAN JOSE	~~3:50 pm~~ 6:55	~~6:11 pm~~ 9:16	econ	
Sat.	11-5-83	AMERICAN	596	SAN JOSE	CHICAGO	9:21 am	3:15 pm	econ	
Mon.	11-7-83	UNITED	316	CHICAGO	BRADLEY	11:29 am	2:21 pm	econ	
	Avis	Rental car is reserved at San		Jose. Driver will take	to and return from	Bradley.			

FIRST CLASS FARE _____ ECONOMY FARE _____ APPROVED BY: R. Naden
(Please type name)

TAX _____ TAX _____

TOTAL _____ TOTAL _____

(Written signature) 10/31

TRAVEL ACCOUNT NUMBER

LOCATION 33 - 36	DEPT. 36 - 39	ACCOUNT 40 - 46		MODEL 47 - 49
404	7725	7368	000	000

TIMEX INTER-OFFICE MEMO

TO: G. Muscio

DATE: October 31, 1983

FROM: L. Galie LG

SUBJECT: DRIVER

DISTRIBUTION:

Please make arrangements to have a driver for Lon Hildreth on Thursday, November 3, 1983 to go to Bradley for a ~~1:45~~ 11:45 pm flight to Chicago on United ~~527~~ 433 425.

Also, to have a driver pick Lon up at Bradley on Monday, November 7th at 2:21 p.m. for the flight from Chicago, United 316. Thanks.

LG:js

Options for CP/M without 32K RAM addition

Put CP/M in ROM in I/F box
    what about addressing?
    can ROM be in lower addr space with RAM above it
    Or use a few bytes at top of RAM that will do
    a "bank switch" to the CP/M ROM.

Put CP/M on wafer - leaves about 24K for programs that don't overwrite FDOS.

Where is the stack set up?

Does CP/M allow multiple files to be open simultaneously? There is only one buffer area. This is the default buffer at 80h.

11/1 am meeting with Mike and Vic re microdrive hdw/software interface on microdrives

Code running in upper 32K must be able to select amongst lower 32K RAM, lower home ROM, and I/F ROM (in some way, shape or form)

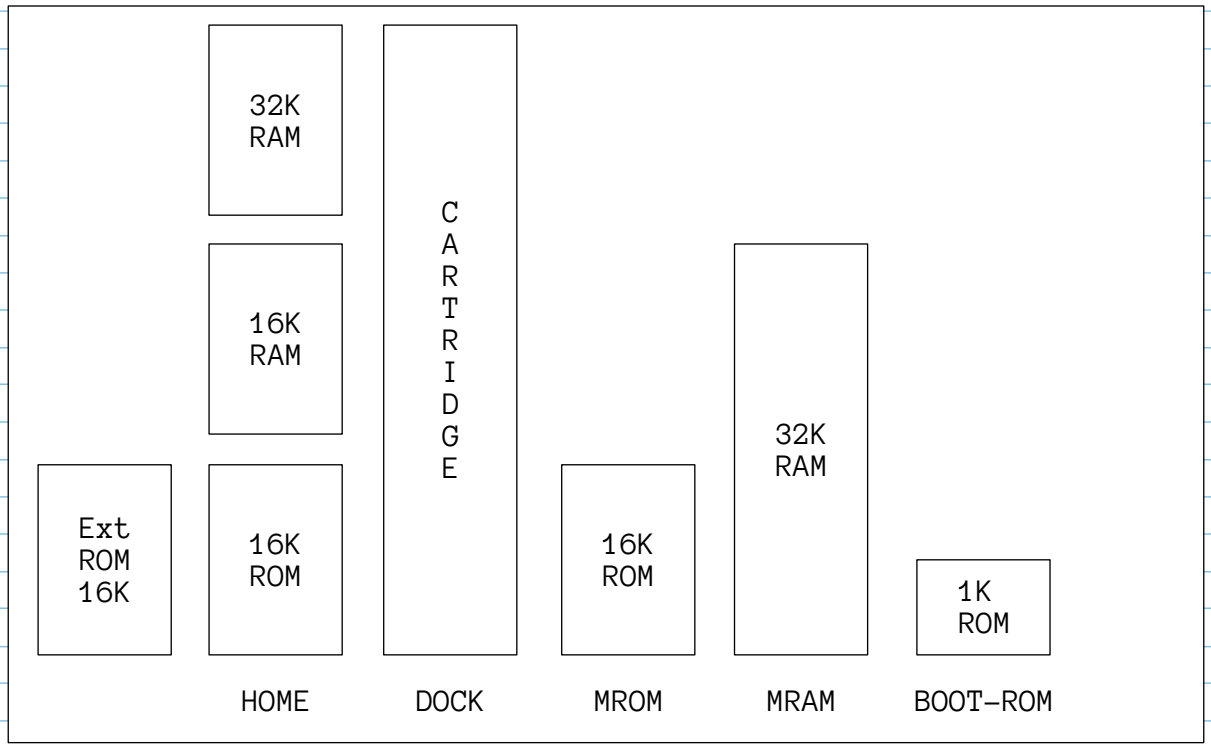
CP/M must handle cartridge software

In CP/M mode, no trapping done on addr 8. Machine comes up w/ 1K Boot ROM selected, and upper 32K of home RAM.

Boot ROM delected via OUT(PORT),BOOTOFF.
Lower 32K RAM selected via OUT(PORT),RAMON. lower 32K of home bank selected via OUT(PORT),HOMEON. MROM selected via OUT(PORT),MROMON. Turning on any of these automatically turns off the others.

If CP/M cartridge software needs RAM 0-100h how can it be an LROS? Can access 0-100h via BIOS (or code in upper 32K RAM). Code in DOCK chunk 7 can't access FDOS. Code in DOCK chunks 0-3 cannot access MRAM directly. Code in DOCK chunks 0 and 1 can't access MROM directly.

Any MROM routines called by BIOS will not have any calls to EXT. ROM.



For BIOS to handle cartridge.

In init routines (GOCPM) port F4 is checked at address 0 and address 8000 for the presence of a cartridge. If there is one to be autostarted, control is passed to its starting address rather than loading the CCP. All initialization up to that point proceeds as normal.

It is the responsibility of the cartrige to handle 32K RAM interaction.

ROM cart cannot go in chunks 0 or 7. Could start at 2000h and go up to (hopefully) 0DFFFh (48K). So 48K ROM is max, using 1F00h (7.75k) bytes RAM in chunk 0.

To change video mode from 40 to 80 col, write a 6 to port FF. To go from 80 to 40, do OUT(FF),0. Can load chr set at 7B00h - just above Dfile2. This keeps it out of the way. 7B00h-7FFFh is otherwise unused.

Possibly FDOS will drop into chunk 6. This would reduce cartridge to 40K ROM, but still has 7.75k + whatever of ch. 6 RAM

11/4 Digital Research

Bruce Arwine
Steve Williams
Steve Cavender (408) 646-5240
secty -6255

99.99% of applications run assuming addr 0.

Visual CCP would not fit starting at 8000. It's about 37k now without helps and some cmds. Will probably get to be ~~240,30K~~

ROMable would be at top of user addr space

Most 8-bit apps require at least 48K RAM (TPA size). Peachtree requires 52K. Tmx would have to convert them all.

Cartridge software would use wafer for read/write

I gave Steve Cavender my TS2068 User Manual.

Personal CP/M

BIOS
    113 - direct screen functions
        direct call to routines to handle VT52 controls
        ptr to control block
        cursor on/off
        alternate keypath mode

    subfunction
        0 - subfunction supported - rets
            bit vector of what's supported
        1 - subfunction emulated
            indicates hardware or software support.

Logically, there are 3 screen planes (combined into frame)
    char, attr, and color
    any rectangle in can be changed
        - COPY
        - ALTER } performed on rectangle

BDOS changes from 2.2

Auto login when a change of media is detected

Block move to console - rather than a char at a time

Error return mode - rets BDOS error to app

Can put dictionary of "known" location - we may want to define a "home" which is where dir goes.

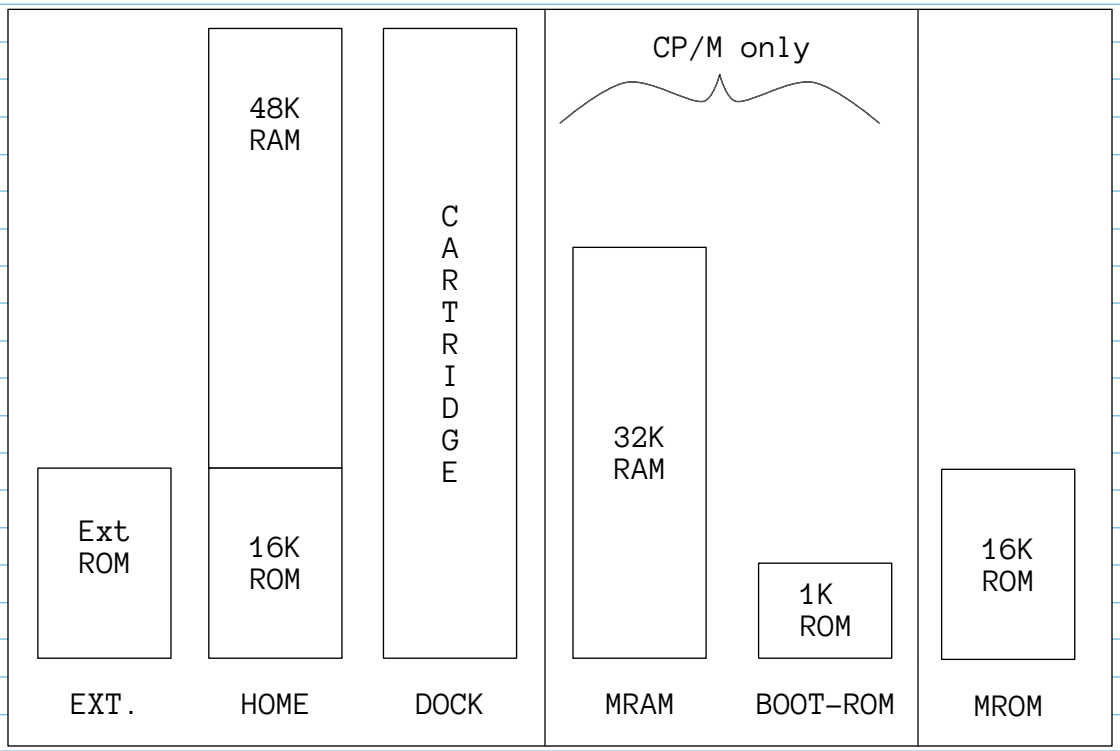
Probably want an app to XFER TS files to CP/M. Use something in order to prevent TMOS from reading CP/M files.

Only need to keep 512 byte buffer for last accessed file (for blocking/deblocking purposes; still has 128 byte buffers for each open file).

CP/M net goes on top of a LAN

Microdrive **11/1/83**
SM

CP/M + TMOS



TRANSFER OF CONTROL	MECHANISM/CONSTRAINTS
1. HOME <=> EXT	N/A in CP/M mode
2. HOME <=> DOCK	via port F4; code in chunk 7 of DOCK cannot access FDOS.
3. HOME <=> M-RAM	via port XX; lower 32K of HOME can't access M-RAM directly
4. EXT <=> DOCK	N/A in CP/M mode
5. EXT <=> M-RAM	N/A
6. DOCK <=> M-RAM	via port XX; lower 32K of DOCK can't access M-RAM directly
7. B-ROM <=> HOME	BROM deselected via port XX in BIOS
8. HOME <=> M-ROM	via port XX; BIOS does not use any M-ROM routines that call EXT. ROM
9. DOCK <=> M-ROM	via port XX; lower 16K of DOCK can't access M-ROM directly; DOCK code in CP/M mode should not call any M-ROM routines that calls EXT ROM.
10. M-RAM <=> M-ROM	via port XX; M-RAM can't access M-ROM directly and should not call any routines that call EXT ROM.
11. EXT <=> M-ROM	N/A in CP/M mode. There is no trapping on address 8 in CP/M mode.

CP/M **11/1/83**
SM

1. When in CP/M mode, trapping on address 08h is disabled.

2. To enable the 32K RAM in the microdrive controller, the value ____ is written to port ____.

3. To enable the 16K ROM in the microdrive controller, the value ____ is written to port ____.

4. The values written to the ports to enable the 32K RAM and 16K ROM in the microdrive contoller make the enabling of both at the same time impossible. It isn't necessary to enforce this in HW. However, at system powerup, both of the memories should be explicitly disabled.

Control codes used by BDOS - decimal value

C - 3	U - 21
E - 5	Z - 26
R - 18	X - 24
H - 8	S - 19
Q - 17	P - 16
I (tab) - 9	

BIOS spec should be ready by 11-11 (Friday)
early week of 11-14 11-18

VCCP src get when contract signed

VIP kit

No additional RAM for CP/M

- Taking into account the display files and RAM needed to interface with the microdrive ROM, a reasonable starting address for CP/M RAM is 8000h.
- The Visual CCP is currently 37K and does not is not yet complete. (Even just counting the 2 display files, it would leave only 36K)
- 99.99% of CP/M applications run assuming RAM starts at address 0. If Timex does not provide RAM at 0, all applications would require modification.
- Most applications for 8-bit machines require at least 48K RAM (TPA size). Peachtree software requires 52K.

ROM version of FDOS requires that its addresses map to chunk 7. ~~Otherwise no problem.~~

ROM/32K RAM on CP/M kbd - FDOS can be overlaid by switching out dock chunk 7, however app would need to switch it back in. Can't really do just a warm start since addr 0 is a jump and that code must be above 100h (probably at very top of a RAM based CP/M)

- Problem here is that there must be overhead bytes at either 0000h or 8000h - which would be RAM. So user would have to do something like OUT(244),050h: GOTO USR (0B00h). Unless still has switch and Boot ROM now selects the dock and jumps there to start.
- Applications couldn't be on cartridge - would have to be on wafer.
- Character set would take up room in the ROM. From wafer can load it from a file into chunk 3 of home RAM (3/4k)

11/8 Sched - CP/M BIOS

~~1. cassette I/O~~
~~2040 output~~
~~40/80 column output~~
~~HOME, SETTRK, SETSEC, LISTST~~

Need to figure out more of the interface between CP/M and microdrive code. Such as putting a dir on the wafer, estabilishing a home, what interplay with microdrive MAP and buffers, etc.

Need to do up a document - S.A.

First cut by Friday.

If we set up designated addresses for each m MAP, there should be not much problem with CP/M as far as maps go.

The one 512 byte buffer most conveniently would go in chunk 7 so as to facilitate I/O to the drive as to not need to be XFER from chunk 0 to 7 to 3 to drive.

This buffer would belong to the last accessed file. When another file is accessed, this buffer gets FLUSH'ed or DISCARD'ed from BDOS, so that's not a problem.

Need to know exact working of CP/M kbd. This has potentially become very non-trivial. Could possible just end up calling K_SCAN and a new routine to check new kbd and code to handle results of both.

CES

For show, could conceivably have CP/M LROS cold start with loader loading "CP/M" from drive or LROS and READ/WRITE routines. Could use Will's Dr. Logo kbd routine. Could have LROS that simply downloads CP/M. Don't have CP/M on wafer.

NEED 32K RAM or else can't show anything.

Then have to do BOOT

NEED BDOS and CCP from DRI

Without 32K RAM, we could take 8000 as bottom of RAM but even CCP would not run

Sched - CP/M BIOS

BSO software on VAX, link to PROLOG	11/8
Cassette I/O 2040 output 40/80 column output SETTRK, SETSEC, SECTRAN, LISTST for 2040	11/11
BIOS spec from DRI	11/14
Design spec circulated	11/18
RS232 I/O SELDSK, HOME TMOS initialization available (with microdrive)	11/23
BDOS source and sample BIOS from DRI Cold start loader (LROS version)	11/30
TMOS access modes available	12/2
READ/WRITE, SETDMA	12/16
FLUSH, DISCARD, MOVE	1/6
32K RAM/Boot EPROM available	1/9
BOOT,WBOOT IOBYTE implementation of CONIN,CONOUT,CONST, AUXIN,AUXOUT,LIST,LISTST	1/13
Centronics output COPY,ALTER	1/20
CP/M kbd available	1/20
kbd input	1/25
CCP src avail	1/25
Project complete	1/31

11/14

What about errors produces by MROM? Will error get passed back to BIOS or will BASIC print it out and get control? Or will it go to the new RAM thinking it's home ROM?

ROMERR - Sinclair code rets to RESET from ROMERR

SH_ERR - after printing shadow ROM err msg, it rets to EDIT, in LED32 after CALL PUTMES

Perhaps we could add a system variable, ERR_RET, that contains an error return address. If it is 0 then returns are made to HOME ROM as before. If it is not 0 then the return is made to that address after the select status of all "banks" is restored.

This check is made before a msg is printed. It is probably made first thing in the RST SH_ERR.

Probably will not generate a ROMERR through CP/M.

Also, in this manner, any application could handle errors any way it wanted.

Load ERR_SP with SP pointing to proper err ret addr, which was pushed on stack. MROM already loads err ret addr from ERR_SP.

SELDSK puts the drive # into DEFDRIVE sys var. Also sets DPH addr

Input (c) contains 0 for drive A - mdrive 1

```
      ~~LD HL,DPBASE~~                LD A,C
      ~~LD A,C~~                      CP 8
      ~~AND A~~                       JR C, ERR ; if drive # >8
      ~~RET Z~~                       INC A
      ~~LD DE, 10h~~                  LD (DEFDRIVE),A
~~NXTDPH:~~                           LD HL,DPBASE
      ~~ADD HL,DE~~                   DEC A
      ~~DEC A~~                       RET Z
      ~~JR NZ, NXTDPH~~               LD DE,10h
                        NXTDPH:       ADD HL,DE
                                      DEC A
                                      JR NZ,NXTDPH
                                      RET
                        ERR:          LD HL,0
                                      RET

                                      29 bytes
```

```
Disk Parameter Header (16-bytes per drive - total 128 bytes)

      0000  0000  0000  0000  ______     _______    _____      _____
      XLT   Scratchpad       DIRBUF     DPB        CSV        ALV
                             Same for   Same for   Diff       Diff
                             all DPHs   all DPHs   16 bytes   31 bytes

DPBASE:
      defw 0      ; translations not done
      defw 0      ; scratchpad
      defw 0
      defw 0
      defw DIRBUF       ; 128-byte scratchpad area for directory ops within
                          BDOS
      defw DPBLK        ; disk param block
      defw CHK01        ; scratchpad - s/w check for changed disk
      defw ALL01        ; scratchpad - keeps disk storage allocation
        •               ;  information. Is this microdrive map?
        •               ; how is this used by BDOS?
        •
      defw 0
      defw 0
      defw 0                          The bits in each byte are backwards.
      defw 0                          eg CP/M keeps sector 0 in byte 1,
      defw DIRBUF                     bit 7. TMOS has it in byte 1, bit 0.
      defw DPBLK                      Sector 1 in byte 1, bit 7 - CP/M
      defw CHK08                           "      byte 1, bit 2 - TMOS
      defw ALL08

Disk Parameter Block
DPBLK:
0BBh  defw  0FFh        ; SPT - sectors per track
      defb  2           ; BSH - data allocation shift factor
      defb  3           ; BLM - data allocation block mark
      defb  0           ; EXM - extent mark
0B0h  defw  0F0h        ; DSM - total storage capacity of drive (in what
                        ; units?)
0AFh  defw  0EFh        ; DRM - total # of directory entries that can be
                        ; stored on drive
0FFh  defb  0FFh        ; AL0 - determine reserved directory blocks
0E0h  defb  0FFh        ; AL1
2Ch   defw  3Ch         ; CKS - size of directory check vector
      defw  0           ; OFF - # of reserved tracks at beginning of
                        ; (logical) disk
```

Questions for Steve Cavender

In the Disk Parameter Block (DPB) ~~what unit is DSM (total storage capacity)? is it~~ # of blocks, ~~# of bytes, etc.~~

Can I use a data allocation size (BLS) of 512 and does that indeed give BSH=2 and BLM=3 and EXM=0?

Does BDOS expect registers to be preserved on calls to BIOS? No.

Where does the stack generally reside and how big is it?

~~Is it standard procedure to have the FCB just before the 128 byte buffer?~~

Is FCB addr still in DE when BIOS gets control?

~~How is the ALL01... in the Disk Parameter Header used by BDOS?~~

In attribute plane of frame,
bit 4,5 = 2 - front of 2 cell char
3 - back of 2 cell char

What is a "2 cell character"? What constitutes normal 8-bit char?

~~Clarification of FRAME structure byte 2~~

In what format are the characters in a FRAME expected to be? ASCII

~~In ALTER RECTANGLE, with what do the rects get AND'd or XOR'd?~~

What sort of info is expected to be returned by subfunction 3: IDENTIFY TERM? What is meant by ALTERNATE KEYPAD MODE (using CP/M kbd) and HOLD SCREEN MODE?

TPA size for most apps is 48K. Any idea how much of that might be ROMable?

Is ~~frame plane~~ 0 assumed to be bit-mapped?

~~What should COPY RECT do if # of cols and rows are not same in SRC and DEST?~~

RS232

Given that the BAUD rate is set, then CONIN and READER will just call IN_BYQ in the MROM. CONOUT, PUNCH, and LIST will just call OUT_T2.

CONST always returns ready.

11-21

```
READN requires
SELDSK - (IX+CHDRIV)            call DRVSEL
                                call RDNXT
                                call WTHDR
                                call RDHBLK - read in CHHDR, HDNUMB,
                                call ECH         HDFLAG, HDNAME, HDCHK
                                call RDBLK  - reads in CHDES and DCHK
                                call ECH
? - (IX+CHNAME)                 call CPSTR
SETSC - (IX+CHREC)
                                call ECD

WRREC
(IX+CHDRIV)                     call DRVSEL
                                call DELBC
                                call WRC
(IX+CHMAP)+MAP                  call SPACE?
                                no space
                                call DELIX - BIOS does not want channel
                                call DRVSEL      deleted. To prevent this,
                                error            BIOS could duplicate code
(IX+CHNAME)                                      of WRREC and WRC and then
(IX+CHBYTE)                                      use MROM for MWR4 and on.
(IX+CHREC)
                                call ECH
                                call ECD
                                call WRFREE
                                call WTHDR
(IX+HDNUMB)                     call FREE?
                                call DELBC
                                call WRBLK
```

File name is in the FCB (File Control Block)

default FCB is at 005Ch

FCB is 33 or 36 bytes

```
Offset
0    :    dr          - drive code (0=use def dr; 1=drive 1, ... 8=drive 8)
1    :    filename    - uppercase ASCII, bit 7=0
9    :    type        - file type (0-3 chars)
12   :    ext         - current extent #
13   :    s1          - internal
14   :    s2          - internal
15   :    rc          - record count for extent "ex" (0-127)
16   :    d           - system use
32   :    cr          - current record to read/write in sequential access
33   :    r           - random record # (0-65535)
```

FCBs are stored in the directory area of the disk. It's brought into memory by OPEN and MAKE, is updated as operations occur and is written back to disk by CLOSE.

FCB address is passed in DE to all BDOS functions requiring it.

Maybe READ/WRITE can access the FCBs.

What about MAPs? How to they interface with BIOS. IS the MAP ALL01 .. ALL08? in DPBASE.

In loading a code file, GETMBL calls CPBLK which copies the record from the buffer to the dest addr. LD BC,(IX+RECLEN) LD DE, (DR_HD_OD), HL=src addr (HL <- IX+CHDATA+9 = beg of DATA)

READN reads record CHREC as pointed to by IX ptr to channel data. BIOS could do its own move to the DMA addr.

In MROM, if I open a file with 128 byte logical records, read record 0, what does MROM do when I then read record 1?

CHBYTE will contain offset to first byte of logical record.

Reading ~~from~~ with blocking/deblocking buffer

Keep a BIOS var indicating whether or not the physical record currently in the buffer has been modified.

```
READ:       LD HL,(IX+CHDATA)
            LD A, (SECTOR#)
            LD C,A                  ; C gets logical record #
            AND 00000011b           ; A gets what will be shifted out
            SRA C
            SRA C                   ; C now has physical record #
            PUSH AF                 ; is this record in the buffer
            LD A,(IX+CHREC)
            CP C
            JR NZ,NOTIN
INBUFFER:   POP AF                  ; here if same physical record
            CP 3
            JR Z, FOURTH
            CP 2
            JR Z, THIRD
            CP 1
            JR NZ, MOVE
            LD (IX+CHBYTE),0        ; here if first logical rec of sector

SECOND:     LD (IX+CHBYTE),128
            LD DE,128               ; here if 2nd
            ADD HL,DE
            JR MOVE
THIRD:      LD (IX,CHBYTE),256
            LD DE,256
            ADD HL,DE
            JR MOVE
FOURTH:     LD (IX,CHBYTE),384
            LD DE,384
            ADD HL,DE
MOVE:       LD DE,(DMADDR)          ; move to DMA address
            LD BC,128
            LDIR
            RET
NOTIN:      CALL READN              ; first check to see if modified what is in
            JR INBUFFER             ; buffer, If so, write it out.
```

```
WRITE:      LD DE,(IX+CHDATA)
            LD A,(SECTOR#)
            LD C,A
            AND 3                ; A gets # of logical records offset from C
            SRA C
            SRA C                ; C gets physical record #
            PUSH AF
            LD A,(IX+CHREC)
            CP C
            JR NZ,NOTIN
INBUF:      POP AF               ; here if same physical record
            CP 3
            JR Z, FOUR
            CP 2
            JR Z, THREE
            CP 1
            JR NZ,MOVE
            LD HL,128            ; here if 2nd
            ADD HL,DE
            EX DE,HL
            JR MOVE
THREE:      LD HL,256
            ADD HL,DE
            EX DE,HL
            JR MOVE
FOUR:       LD HL,384
            ADD HL,DE
            EX DE,HL
MOVE:       LD HL,(DMADDR)
            LD BC,128
            LDIR
            LD (BUFMOD),1
            RET
NOTIN:      LD A,(BUFMOD)
            CP 1
            JR NZ,READITIN
            CALL WRTASECTOR
READITIN:   CALL READN           ; read in whole sector of one to be
            JR INBUF             ; written to
```

TS screen data

Scan line 1 of rows 0–7, then line 2 of rows 0–7, etc.

Attribute format

- 7 FLASH
- 6 BRIGHT
- 5
- 4 PAPER
- 3
- 2
- 1 INK
- 0

stored as row 1, row 2, ... , row 24 (32 bytes each)

CP/M

Plane 1: Attributes

bit
0,1 - don't care
2   - don't care (INVERSE in software)
3   - BRIGHT
4   - don't care (can underline by setting buts in line 8 of char)
5   - FLASH
6,7 - don't care

Plane 2: Color plane

In 80 col mode there is no BRIGHT or FLASH and only one color.

In 40 col mode, 4 characters go into 3 bytes, so attrs and color apply
to either intended char and 1/3 of next
to either intended char and all prev char and 2/3 of next
to either intended char and 2/3 of prev and all of next
to either intended char and 1/3 of prev

May just say that planes 1 and 2 don't exist (indicated by FRAME structure)
but INVERSE and underline are still possible in software.

```
FRAME:
      defw  PLANEPT
      defb  1           ?
      defw  40          ; # of columns
      defw  24          ; # of rows

PLANEPTR:
      defw  4000h
      defw  0
      defw  0           ?

RCB structures
      defw  FRAME       - 0 for console
      defw  Origin column
      defw  Origin row
      defw  # of cols
      defw  # of rows
```

Is there some sort of CreateFrame function?

```
4000        DFile1
5AFF
5D09        sys.vars

6000        DFile2
7AFF
7D00        CH_SET
7FFF
```

What format of scan lines is screen expected to be in?

There may be many FRAMEs - can then use COPYR to provide animation. COPY may do translation into proper mode if COPYing between 0 ptr to a non-0 ptr. And consider 40/80 col mode. And consider normal or graphic char. In graphic mode, copy it bit by bit. In normal mode do translation. Well...

Take 80 column mode rectangle on FRAME=0 (screen) to be working on both display files.

1920 (780h) bytes for "attributes" of 80 column screen. This is almost 2K. Is it worth it?

The fact that the Allocation Vector is in a different format than the microdive MAP may not be a problem. The MAP is accessed only by the MROM and the AV is accessed only by the BDOS. Just have to make sure a MAP exists. How is this done?

> Create creates a map at (CHANS) and marks it full.
> OPENF goes through and determines the free sectors.

May not even need to use MAPs - wrong because MWR3 calls WRFEE to write to the next free block.

Could possibly duplicate all of WRC and WRFREE such that it doesn't use MAP.

BDOS has already selected what sector to write to.

Call WTHDR until find proper one as in WRSC1 in SH_HOOK. So in reality, duplicate WRSC of SH_HOOK omitting the marking of the MAP (0110-0157).

This just calls ECH, ECD, WTHDR, DECSEC, DELBC, and WRBLK.

Turn on the drive first.

Clarification of byte 2 of FRAME structure.

Bit map of plane usage.
     If bit=1, then a real plane is needed
     If bit=0, then one byte is used for every element of the plane.

E.g. our screens can have the 8 different color combinations via this phony color plane.

value	paper	6+8*colorpaper
6	black	
14	blue	OUT (FF),val
22	red	
30	magenta	
38	green	
46	cyan	
54	yellow	
62	white	

Carol suggests that graphics in 40-col mode be allowed to use attributes.
Can't enforce no attrs on chars while allowing them for graphics.

So do graphics in 32 col mode?

r1-1 for rows 0-7
r2-1 st=row*32+col+16384
r3-1
r4-1 for rows 8-15
r5-1 st=(row-8)*32+col+18432
r6-1
r7-1 for rows 16-23
r8-1 st=(row-16)*32+col+20480
r1-2
r2-2
  •
  •
  •
r8-8
r9-1

BIOS routines 100% written

SETTRK
SETSEC
SECTRAN
HOME
~~LISTST~~
RS232 I/O

32K IOBYTE portion of CONIN, CONOUT, CONST, AUXIN, AUXOUT, LIST, LISTST
CONST for RS232 and cassett input

What is expected date for VIP kit?

Committed to on 1-17-84

If init MROM by FW4

Cassette and mdrive BIOS ready (for LOGO) FW6

IF 32K RAM by FW8

BIOS ready by FW12 (for BDOS and CCP)

Ready for QA by FW14

BIOS routines 90% written

Cassette I/O
40/80 column/2040 output
READ/WRITE/FLUSH (TMOS)
SELDSK
COLD START LOADER - in BOOT ROM

Not done at all
SETDMA
~~FLUSH~~
DISCARD
MOVE
RECTCOPY
RECTALTER
kbd input - CP/M kbd
Centronics - TMOS
CONST for kbd - CP/M kbd
CONOS
AUXIS
AUXOS
DSCRF - Direct Screen Functions
BOOT/WBOOT - 32K RAM

CP/M BIOS SCHED 12/14/83

Routines 100% written but not tested

SETTRK, SETSEC, SECTRAN, HOME, LISTST, RS 232 I/O
IOBYTE portion of CONIN, CONOUT, CONST, AUXIN, AUXOS, LIST, LISTST
CONST for RS232 and cassette input

Routines 90% written (need to know what value written to what port)

Cassette I/O
40/80 column/2040 output
Cold start loader
READ/WRITE/FLUSH
SELDSK

	Cassette I/O
1/20	40/80 column/2040 output
1/23	TMOS initialization available (with microdrive)
2 wks	SELDSK READ/WRITE/FLUSH COLD START LOADER (LROS version of BOOT EPROM not avail)
2/16	Cassett I/O and read/write stuff committed to LOGO
3 wks	Direct Screen Functions COPY, ALTER
2/27	32K RAM available
	BOOT,WBOOT
3/2	IOBYTE implementation
3/6	DISCARD, MOVE
1 wk	CP/M kbd avail
3/21	kdb input/CONST for kbd
3/22	Centronics output
1 wk	Integration with BDOS and VCCP
3/30	Project complete

Dr. Logo **1/18/83**

Simple char I/O coding	1/19	Will
Adv char I/O coding	1/23	Carol
Compile and link memory routines	1/25	Will
Simple char I/O debug	1/31	Will
Advanced char I/O debug	1/31	Carol
Cassette BIOS routines ready	2/6	
udrive BIOS routines ready	2/6	
Cassette I/O coding	2/7	Will
Printer I/O coding	2/7	Carol
I/F1 and printer avail	2/7	
Printer I/O debug	2/14	Carol
Cassette I/O debug	2/15	Will
udrive I/O coding	2/21	Carol
I/F 1 and udrive available	2/21	
udrive I/O debug	2/29	Carol
graphic I/O coding	3/5	Will
graphic I/O debug	3/22	Will
Make ROMable	3/29	Will
Project complete	3/29	

Personal CP/M **1/18/83**

Spectrum I/F 1 and udrives for BIOS testing	1/16	FW3
Cassette and udrive BIOS ready	2/6	FW6
TS2000 I/F 1 (full) including 32K RAM	2/20	FW8
BIOS ready	3/19	FW12
BDOS and VCCP ready	3/19	FW12
CP/M utilities ready	3/19	FW12
Project complete	4/2	FW14

TMOS **1/18/83**

TS 2000 I/F 1, 16K EPROM microdrives	1/23	FW4
(ie proper address decoding)		
Implementation and Testing (completion)		
1) Initialization	1/23	FW4
2) Access Modes	1/30	FW5
3) CAT and ERASE	2/13	FW7
Centronics tester avail	2/13	FW7
4) Centronics I/F	2/27	FW9
TS 2000 I/F 1, 16K EPROM Centronics I/F, network/ serial I/F, microdrives (2 sets)	2/27	FW9
5) Miscellaneous		
Copy 80 command	3/12	FW11
Project complete	3/12	FW11

Carol's ADL stuff

IO40 528h = 1.29K TV40 1BDh = 0.43K
IO80 57Eh = 1.37K TV80 215h = 0.52K
IO64 515h = 1.27K est TV64 0.42-0.48K

Unacceptable.

	H	K
CBTABLES	207	0.51
IOBYTE	69	0.10
RDWR	153	0.33
SELDSK	29	0.04
SETS	10	0.02
TAPEIO	6C	0.11
TS2040DR	11	0.02
TV40	1BD	0.43
TV80	215	0.53
	84B	2.07
SCREEN	D9	0.21
COPY40	6E2	1.72
	1006	4.01
PAGE ZERO	100	.25
BIOS DATA	25	.04
(de)blocking buffer	253	.58
	137E	4.87

To add in 32 and 64 col mode, could alter IOBYTE so that AUXIN is 1 byte and CON is 3 bytes and

CON = 4 = 32 col
CON = 5 = 64 col

could maybe even let CON = 6 = high res color
(I don't think CP/M will allow for this - only one color per char)
and theoretically CON = 7= dual screen mode

For space considerations, we may want to keep only the current IOBYTE drivers in memory - loading off mdrive when changed.

MROM address I need

- ~~DRVSEL~~
- WRBLK
- WTHDR
- ECH
- ECD
- READN
- GET_HD
- GETMBL

Sys vars

- ~~ROMADDR~~
- ~~SECTOR~~
- LOADADDR

Can probably use LIN_BU routines to assist in the transfer to/from plane 0 - ~~need to know what the format of line #s is~~ B=line #, C=col#

L = in effect (B mod 8)*32 H = in effect 64+8*INT(B/8)

- 1st line = 0
- 2nd line = 20h
- 3rd line = 40h
- 4th line = 60h
- 5th line = 80h
- 6th line = A0h
- 7th line = C0h
- 8th line = E0h

- upper 1/3 = 40h
- middle 1/3 = 48h
- lower 1/3 = 50h

This gives addr of line B col 0 in HL in 80 col mode, this is always in D-File 1.

The only mode where this isn't true is in dual screen mode where DFile2 needs to add 2000h to result.

Can then add col to get appropriate addr.

Scan lines are 100h apart, ie. scan line 1 of row 5 is at 4080. Scan line 2 of row 5 is at 4180.

Need routine to copy rectangle row, col in 40 and 80 col mode.

Need translation of 40 col (row,col) into physical address. 4 different conditions at left and right edge.

~~LNBU can give starting addr of row~~

```
call CALCPOS in 40-col mode ADL
      INPUT: BC=row/col
      OUTPUT: HL=DFile addr
                 (DFBIT)=bitpos(7,5,3,1)
```

~~Determine starting position of col.~~

```
~~      CALL LNBU~~
~~      LD A,C~~
~~      ADD A,A~~
~~      ADD A,C            ; A = 3*<40 col pos>~~
~~      PUSH AF~~
~~      SRL A~~
~~      SRL A              ; A = A/4 = <32 col pos>~~
~~      LD E,A~~
~~      LD D,0~~
~~      ADD HL,DE          ; DE contains offset into line~~
                           ; HL = starting addr of char
~~      POP AF~~
~~      AND 3~~
~~      JR Z,~~
```

CALCPOS does all this

when caling, make sure (LINLEN)=40 and (MARGIN)=0 or (LINELEN)=80 in 80 col CALCPOS

Meeting Mon 1-16 with TABOR - 3 1/2" disk drive

Dick Sunday

B	C	OUTPUT	HL	DE
~~10~~	~~5~~		~~4800~~	~~807~~
~~10~~	~~4~~		~~5800~~	~~1~~
~~23~~	~~39~~		~~50A0~~	~~F507~~
~~0~~	~~0~~		~~5800~~	~~1~~
OA	5		48C0	E07
OA	5		48DB	1
0	0		581E	1
17	27		4021	2
18	28		4000	1
16	28		4040	1
15	28		4060	
14	28		4080	
13	28		40A0	
12	28		40C0	
11	28		40E0	
10	28		4800	
0F	28		4820	
0E	28		4840	
0D	28		4860	
0C	28		4880	
0B	28		48A0	
0A	28		48C0	
09	28		48E0	
08	28		5000	
07	28		5020	
06	28		5040	
05	28		5060	
04	28		5080	
03	28		50A0	
02	28		50C0	
01	28		50E0	
0			5800	

18	(o) 28	4000
18	27	4001
18	26	4002
	25	4003
	24	4003
	23	4004
	22	4005
	21	6
	20	6
	1F	7
(OA)	1E	8
	1D	9
	1C	9
	1B	A
	1A	B
	19	C
	18	C
	17	D
	16	E
(13)	15	F
(14)	14	F
(15)	13	10
	12	11
	11	12
	10	13
	E	14
	D	15
	C	16
	B	16
	A	17
	9	18
	8	19
	7	19
	6	1A
	5	1B
	4	1C
	3	1C
	2	1D
(27)	1	1E
	0	1E

2.0 Supports cassette - 1.0 does not
microdrive BIOS needed for LOGO

DRI Personal CP/M due March 15.
Documentation due Feb 13

~~LOGO BIOS must duplicate all MROM routines if uses since LOGO must run without the interace unit.~~

What a foolish statement since microdrives are required.

A possible utility - SYSGEN type thing that would update the default IOBYTE. Could read in the CP/M file and change the byte that gets loaded into IOBYTE.

If different I/O drivers are not all always in RAM, initialization would have to check IOBYTE before loading in the default drivers.

2 cell chars primarily used with KANJI chars where they take 80 cols and make it 40 since KANJI chars take up 2 chars.

BIOS saves SP when entered and uses its own stack.

CCP has room for 31 or 32 pushes.

COPY RECT must make transformation from FRAME 0 to other frames and vice versa. Carol's ADL must handle this.

Port E7 controls entrance and exit from MROM and 32K RAM.
bit 0 - MROM
bit 1 - RAM

WordStar
    baby model $49
    full blown higher, bundled

40 col mode would wrap around

currently designed for 8x64 or 16x64

Financial Planning package

runs 8086 CP/M, MS-DOS, PC-DOS

Ctrl-Y to get out of anything escape
Ctrl-U throws away current line
Ctrl-R retype line
Ctrl-Z ends input stream

delete/confirm <filespec>
set password
help

wildcard - % - match exactly one char

File protection - read write execute delete
    system
    owner
    group
    world

e.g. set protection = (s:rwed,o:rwed,g:r,w:)
    system and owner can do anything, group can read, world won't know
    it exists
    set prot = w:e <filespec> - lets world execute file

Create sub director - create/dir [.DOCS]
    create/dir/version_limit=2[.DOCS] limits files in .DOCS to 2 version

edtini.com file with set mode change line will put EDT into screne mode

To replay edit session - edit/recover nome.typ

Rquest "m" sends message to operator's console

MAIL
> help   deletes most recently received msg first

SHOW USERS

PHONE user - converse mode
PHONE answer

LOGIN.COM in home dir
    can set up shortened cmds
    eg PU:==PURGE/KEEP=3

@ executes a cmd file
    cmds in cmd file are preceded by '$'
    eg $set def [.memo] sets default sub directory

Cmd Procedures
1) To execute @
2) To debug
    $set verify
3) $set NOON
    causes proc to continue executing on error.

Logical Name	Default
SYS$INPUT	cmd proc being executed
SYS$OUTPUT	Terminal
SYS$ERROR	Terminal

These can be redefined by $ASSIGN bg new

Assign a symbol
$sym name [:][=]=expr or char <string|arith>
: converst to upper case
= makes global - otherwise it's local to the cmd proc

Arith	expressions	Value
	A=5/3-1	A=0
	A=2+"3"	A=5
	A="2+3"	A="2+3"
	A="2"+"3"	A="23"
	A=2+3	A=5
Strings	FILE="TEXT"+".DAT"	"TEXT.DAT"
	NAME="A.DAT"-".DAT"	"A"
Cmp	A="AB" .LTS. "BA"	A=1
	A="ab" .LTS. "AB"	A=0

Passing symbols

```
      $FORTRAN 'FILE'
      $LINK 'FILE'
      $RUN  'FILE'
```

1) Make a global symbol assignment before executing

```
      $FILE :== PAYROLL
      $@payroll
```

2) Passing a value to the symbol at exec time

```
      $@payroll PAYROLL
            'FILE' is changed to 'P1'
      There are 8 params 'P1' - 'P8'
```

3) Soliciting a value from term or fole at exec time

```
      $INQUIRE FILE "Enter filename:"   - from term
      from file
            $OPEN/READ LIST DIR.LIS    (LIST logical file name)
            $READ LIST FILE
```

```
$PASCAL/DEBUG prog to get symbol table
$LINK/DEBUG prog
$RUN prog

or $RUN/DEBUG prog but no symbols
```

1. Architecture design of initialization of TS 2068

2. Software design of initializaiton and channels

3. Implementation/debug

4. Made several software cartridges for both production cartridges and demonstration cartridge.

5. Wrote Test Spec for Rev 5.0 concentrating on BEU.

6. Assisted Richard Benjamin in debuggin system software for BEU - led to abandoning BEU

7. Studied Sinclair microdrive for feasibility of implementing TDOS and CP/M

8. Design of CP/M BIOS for microdrive, including input on design of interface design.

Timex

Designed architecture of initialization portion of TS2068.
Software design of initialization and channels of TS2068.
Implementation/debug of same.
Studied Sinclair Microdrive code to determine software feasibility of adapting it to TDOS and using it in conjunction with CP/M.
Design and coding of CP/M BIOS.

Table 1-1: Debugger Command Abbreviations

@file-spec	EXAMINE /BYTE /WORD /LONG /QUADWORD /OCTAWORD /FLOAT /D_FLOAT /G_FLOAT /H_FLOAT /ASCII:n /INSTRUCTION /OCTAL /DECIMAL /HEXADECIMAL /SYMBOL /NOSYMBOL /SOURCE
CALL	EXIT
CANCEL ALL	GO
CANCEL BREAK /ALL	HELP
CANCEL EXCEPTION BREAK	SEARCH /ALL /NEXT /IDENTIFIER /STRING
CANCEL MODE	SET BREAK [DO /AFTER
CANCEL MODULE /ALL	SET EXCEPTION BREAK
CANCEL SCOPE	SET LANGUAGE
CANCEL SOURCE /MODULE	SET LOG
CANCEL TRACE /ALL /BRANCH /CALL	SET MARGIN
CANCEL TYPE/OVERRIDE	SET MAX_SOURCE_FILES
CANCEL WATCH /ALL	SET MODE
CTRL/C	SET MODULE /ALL
CTRL/Y	SET OUTPUT
CTRL/Z	SET SCOPE
DEFINE	SET SEARCH
DEPOSIT /BYTE /WORD /LONG /QUADWORD /OCTAWORD /FLOAT /D_FLOAT /G_FLOAT /H_FLOAT /ASCII:n /INSTRUCTION /OCTAL /DECIMAL /HEXADECIMAL	SET SOURCE /MODULE
EVALUATE /OCTAL /DECIMAL /HEXADECIMAL	SET STEP
EVALUATE/ADDRESS /OCTAL /DECIMAL /HEXADECIMAL	SET TRACE /BRANCH /CALL

ZK-017/1-82

Table 1-1 (Cont.): Debugger Command Abbreviations

SET TYPE /**O**VERRIDE	**SH**OW **SO**URCE
SET WATCH	**SH**OW **S**TEP
SHOW **B**REAK	**SH**OW **T**RACE
SHOW **C**ALLS	**SH**OW **TY**PE /**O**VERRIDE
SHOW **LA**NGUAGE	**SH**OW **W**ATCH
SHOW **LO**G	**S**TEP /**INS**TRUCTION /**L**INE /**INT**O /**O**VER /**S**YSTEM /**NO**SYSTEM /**SO**URCE /**NOSO**URCE
SHOW **MAR**GIN	
SHOW **MAX**_SOURCE_FILES	
SHOW **M**ODE	
SHOW **MODU**LE	
SHOW **OU**TPUT	
SHOW **SC**OPE	**T**YPE
SHOW **SEA**RCH	

ZK-017/2-82

Table 8-1 Major Symbolic Debugger Commands

Feature	Description	Command Format
Display values	Display variable contents using symbolic names	EXAMINE variable
Change values	Modify variable contents	DEPOSIT variable = value
Define symbols	Define symbolic names for later use	DEFINE symbol = value
Calculate values	Compute expressions using symbolic names	EVALUATE expression
Get help	Get help for any command	HELP [command_name]
Breakpoints	Suspend program execution at a specified point	SET BREAK %line #
Tracepoints	Monitor order of execution of program lines	SET TRACE %line #
Watchpoints	Suspend program execution when the content of a variable changes	SET WATCH variable (global only)
Test subroutines	Call and pass arguments to a subroutine	CALL sub_name [(arg,...)]
Execute program	• from a given point	GO [address]
	• for a specified number (n) of instructions or lines	STEP [n]
Debug routines	Make symbols from specified module available to debugger	SET MODULE module
	Define default module name for setting tracepoints and watchpoints on symbols whose names appear in more than one module	SET SCOPE module
Stop debugger	Leave debugger and return to DCL prompt	EXIT

```
WORK:[DORSEY.LEVEL1.LIST]PROPAS.LIS;1

Listing of main program and subprogram:

0     LINE      LEVEL
     NUMBERS   PROC STMT  STATEMENT.

          1     1    0   PROGRAM Grades (Course, INPUT, OUTPUT);
          2     1    0
          3     1    0   TYPE
          4     1    0     Yes_no = (yes,no);
          5     1    0   VAR
          6     1    0     Student_name : PACKED ARRAY [1..40] OF CHAR;
          7     1    0     Course : TEXT;
          8     1    0     Icount : INTEGER;
          9     1    0     Done : Yes_no;
         10     1    0     Grade, Total, Average : REAL;
         11     1    0     St, Av : PACKED ARRAY [1..10] OF CHAR;
         12     1    0
         13     1    0   VALUE
         14     1    0     St := 'Student:  ';      Av := 'Average:  ';
         15     1    0
         16     2    0   PROCEDURE Compute;
         17     2    0     BEGIN
         18     2    1       Icount := 0; Total := 0;
         19     2    1         REPEAT
         20     2    2           WRITE (' Input grade (or 0 to end input); ');
         21     2    2            READ (Grade);
         22     2    2              IF Grade <> 0
         23     2    2                 THEN Icount := Icount + 1;
         24     2    2
         25     2    2              Total := Total + Grade;
         26     2    2         UNTIL Grade = 0;
         27     2    1         Average := Total / Icount;
         28     2    1       END;
         29     2    0
         30     1    0   BEGIN  { Main Procedure }
         31     1    1
         32     1    1    REWRITE (Course);
         33     1    1     REPEAT
         34     1    2     { Get information for one student }
         35     1    2              WRITELN;
         36     1    2              WRITE ('Student name? ');
         37     1    2              READ (Student_Name);
         38     1    2              Compute;
         39     1    2     { Output results to terminal and file }
         40     1    2              WRITELN;  WRITELN;
         41     1    2              WRITELN (St, Student_Name, Av, Average :3:1);
         42     1    2              WRITELN (Course, St, Student_Name, Av, Average :3:1);
         43     1    2     { Check if more students }
         44     1    2              WRITELN;
         45     1    2              WRITELN ('Are you done ? (Yes/No)  ');
         46     1    2              READ (Done);
         47     1    2       UNTIL Done = Yes;
         48     1    1     CLOSE (Course);
         49     1    1   END { Program Grades }.
```